Unify DataServer™ /ELS

Developer's Reference

UNIFY

© 1987, 2000, 2003 Unify Corporation. All rights reserved.

No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise without the prior written consent of Unify Corporation.

Unify Corporation makes no representations or warranties with respect to the contents of this document and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Unify Corporation reserves the right to revise this document and to make changes from time to time in its content without being obligated to notify any person of such revisions or changes.

The Software described in this document is furnished under a Software License Agreement. The Software may be used or copied only in accordance with the terms of the license agreement. It is against the law to copy the Software on tape, disk, or any other medium for any purpose other than that described in the license agreement.

The Unify Corporation Technical Publications Department values and appreciates any comments you may have concerning our products or this document. Please address comments to:

Product Manager
Unify Corporation
3927 Lennane Drive
Sacramento, CA 95834-1922
(800) 24-UNIFY • (916) 928-6400 • FAX (916) 928-6401

Unify DataServer, Unify eWave, Unify VISION, Unify WebNow! and the Unify logo are trademarks of Unify Corporation in the United States and other countries. Unify and ACCELL are registered trademarks of Unify Corporation in the United States and other countries. Other brand or product names shown are trademarks of their respective owners. Printed in U.S.A.

Part Number: 7061-02

Contents

About this manual 13
Reference Manual Organization **13**
Syntax Conventions **17**
Other Conventions **17**

Part I: The Unify DataServer/ELS environment 19

Chapter 1: **Introduction to Unify DataServer/ELS 21**
Building a Unify DataServer/ELS database application **23**
References **24**

Chapter 2: **Unify DataServer/ELS and the operating system environment 27**
File name suffixes **28**
Directories and path names **28**
Text editors **29**
The print spooler **29**
Running operating system commands from Unify DataServer/ELS **30**
Using operating system security from Unify DataServer/ELS **31**

Chapter 3: **The Menu Handler 33**
Standard Unify DataServer/ELS menus **33**
Users **39**
Help documentation **40**
Executables **41**
Starting Unify DataServer/ELS automatically **41**
Customizing Menu Handler commands **42**

Chapter 4: **Files and directories 45**
Files **46**
Directories **49**

Chapter 5: **Environment variables 53**
Using environment variables **53**
Setting environment variables **53**

Unify DataServer/ELS environment variables **54**
Environment variables and Unify DataServer/ELS programs **67**

Chapter 6: **Termcap and Unicap 69**
Termcap **70**
Terminal functions **70**
Optional terminal functions **73**
Keyboard mapping **73**
Setting up a termcap file **74**
UNICAP **75**
Character classes **75**
The standard unicap file **79**

Part II: Database design and maintenance 85

Chapter 7: **Introduction to database design 87**
Database elements **89**
Field data types **89**
Refining the database design **95**
Eliminating multiple values in fields **95**
Eliminating non-key field dependencies **98**
Database diagrams **104**
Data access methods **106**
Hashing (primary key) **107**
Explicit relationships **108**
B-trees **110**
Buffered sequential access **115**

Chapter 8: **Database design utilities 119**
Database directory structure **120**
Design and create a new database (dbcreate) **121**
Database tables **121**
Database fields **124**
Creating the database **131**
Modify database design (schent) **136**
Modifying values on the table entry form **137**
Modifying values on the field entry form **140**
Create database (crdb) **145**

Database files **145**
Reconfigure database (scom) **148**
Preparing to reconfigure **149**
Reconfiguring **150**
Finishing the reconfigure **152**
Recovering corrupted hash table fields **153**
Add, drop B-tree indexes (idxmnt) **156**
Inquiring, rebuilding, and dropping B-trees **159**
Adding B-tree indexes **161**
Database field entry prompts **161**
Processing B-tree definitions **162**
Advanced field attributes (afa) **164**
AFA file structure **167**
Command syntax **168**
An example file **178**

Chapter 9: **Database maintenance utilities 181**
Write database backup (budb) **182**
Read database backup (redb) **184**
Reading the database backup **185**
Replaying the transaction log **186**
Rebuild the hash table (rekey) **191**
Rebuild explicit relationships (repoint) **194**
From the Menu Handler **195**
From the operating system **197**
Define database volumes (volmnt) **200**
Using a raw disk partition **202**
Using ordinary operating system files **203**
Defining database volumes **205**

Part III: System administration 213

Chapter 10: **Database security utilities 215**
Modify system parameters & security (parmnt) **216**
Add or modify group privileges (grpmnt) **218**
Add or modify individual privileges (empmnt) **222**
Field security maintenance (fldsec) **227**
Process field passwords (procpass) **230**

Using the operating system security **231**

Chapter 11: **Data Dictionary reports 233**
Print database design(schlist) **234**
Print menus (lmenu) **241**
Print screens (sfrep) **244**
Print group privileges (lgrp) **245**
Individual access listing (lemp) **247**
Print help documentation (prthdoc) **249**
Print list of programs (lexec) **250**
Print, display database statistics (dbstats) **252**
Display hash table statistics (hts) **255**
Print, display B-tree statistics (btstats) **257**

Chapter 12: **Miscellaneous utilities 259**
Transaction logging **260**
How transaction logging works **260**
Transaction logging status (txconf) **262**
Add, modify or delete menus (menumnt) **266**
Describe program to system (execmnt) **269**
Create or modify help documentation (enthdoc) **273**
Edit SQL or RPT command files **275**
Database load (DBLOAD) **276**
Program loading (lfilegen) **280**
Monitoring locks (lmpeek) **281**

Part IV: Screen form development and maintenance 283

Chapter 13: **Screen form development tools 285**
Create default screen form(cdsf) **289**
Paint screen forms (paint) **293**
Screen editing commands **295**
Customizing PAINT commands **308**

Chapter 14: **Screen form maintenance utilities 313**
Check screen form coordinates (sfmaint) **314**
Display list of screens (sfslist) **319**
Test screen (sfsamp) **320**

Compile screens (sfproc) **321**
Restore screen to Data Dictionary (sfrestr) **322**

Part V: Data entry and retrieval 323

Chapter 15: **ENTER–data entry query by forms 325**
The Rules of Using ENTER **326**
Register screen form with ENTER (entmnt) **329**
Using ENTER for data entry **335**
Entering the primary key field **335**
Moving the cursor on the ENTER screen **335**
Adding or changing field data **336**
Entering data in a text type field **336**
Enter data in a binary type field **337**
Changing data in a secondary table **337**
Using ENTER with advanced field attributes (AFA) **338**
Adding and modifying records with AFA **338**
ENTER and AFA messages **339**
Using ENTER for query by forms **340**
Inexact matching on string fields **341**
Inexact matching on non-string fields **342**
Using queries to modify, delete, or report **343**
Enter report options screen **344**

Chapter 16: **SQL-Query/DML language 349**
Query facilities **351**
Required and optional clauses **351**
Identifying tables and fields in queries **352**
Entering query statements **352**
Help features **353**
select...from CLAUSE **355**
where CLAUSE **358**
unique KEYWORD **366**
Arithmetic expressions **367**
Aggregate functions **369**
order by CLAUSE **371**
group by CLAUSE **372**
Nested queries **375**

having CLAUSE **378**
Join queries **381**
Variable Queries **388**
Data Manipulation Language facilities **390**
insert statement **390**
update statement **393**
delete statement **394**
Unify DataServer/ELS SQL extensions **396**
Editing queries **396**
Creating and running SQL scripts **397**
Running operating system commands **398**
Dumping data to external files **399**
Loading data from external files **400**
Using text fields with SQL **404**
Using binary fields with SQL **405**
Using SQL internal tables and environment variables **405**
SQL/Screen Form/Report interface **413**
SQL by forms (ssql) **413**
Registering a screen form with SQL (ssqlmnt) **415**
Using an SQL screen **421**
Using the SQL report option screen **423**
SQL reference **427**
Keyword summary **427**
Formal language syntax **427**
Query statements and clauses **428**
DML statements and clauses **441**
Non-Query commands **444**
HELP commands **448**
SQL errors **449**

Part VI: Report generators 451

Chapter 17: **RPT–Report processor 453**
Report writing concepts **454**
Files used by RPT **454**
Example report **455**
Basic report examples **459**
Department listing example **459**

Form letter example **469**
Advanced report techniques **475**
RPT expression syntax **486**
Named expressions **488**
Logical expressions **489**
Expression components **493**
Report fields **493**
Numbers, dates, and times **494**
Strings **499**
Functions **505**
Variables **510**
Expression validity table **513**
Control break processing **514**
Named sort expressions **514**
Control breaks **517**
Control break processing command-groups **518**
Non-command-groups **520**
bottom margin **520**
end **520**
input **521**
left margin **522**
length **522**
separator **522**
sort **523**
top margin **524**
width **524**
Command-groups **525**
after report **525**
after name **526**
before report **527**
before name **527**
detail **528**
header **529**
Command-group commands **530**
if **530**
need **531**
page **532**

print **532**
set **540**
skip **540**
Using RPT with other Unify DataServer/ELS tools **541**
SQL by forms and RPT **541**
ENTER and RPT **543**
rip **544**
User programs and RPT **544**
RPT table usage information report **546**
RPT reference summary **552**
RPT keywords **552**
Overview of commands and command-groups **552**

Chapter 18: **LST–Listing processor 557**
LST Syntax conventions **558**
Expressions **558**
Selecting records **560**
Running the selection processor **560**
Selection processor commands **560**
Listing records **564**
Running the listing processor **564**
Listing processor commands **564**
sort **568**
total **569**
go **570**
print **570**
nohead **571**
unlock **571**

Appendix A: Field and internal data types 573
Changing field data types and lengths **574**
Safe Changes **575**
Unsafe Changes **576**

Appendix b: Unify DataServer/ELS reserved words 579

Appendix c: Custom message control 581
Custom message control **582**

Formatting currency and date displays **583**
Reconfigure data dictionary **584**
What is a data dictionary? **584**
Customizing the data dictionary **586**

Appendix D: Unify DataServer/ELS messages 597

Appendix E: The licprod utility 685
Licensing after installation **685**
Licensing during installation **687**

Appendix F: Tuning your operating system 691

About this manual

This is the *Unify DataServer/ELS Developer's Reference Manual*. The manual provides specific information on how to use each Unify DataServer/ELS tool. The index at the back of the manual is particularly useful for locating a specific subject.

While some chapters in this manual give examples of use, the *Unify DataServer/ELS Developer's Tutorial Manual* provides more examples of how most Unify DataServer/ELS tools are used to develop an application system. It is the best place to start if you have never used Unify DataServer/ELS before.

The *Unify DataServer/ELS Developer's Reference Manual* assumes the following:

- You know how to use your computer's operating system.
- You know what a database management system is.

If you are already familiar with database systems and want to see how to program using Unify DataServer/ELS, refer to the *Unify DataServer/ELS Direct HLI Programmer's Manual*.

Reference Manual Organization

The *Unify DataServer/ELS Developer's Reference Manual* consists of 18 chapters, grouped in six parts, followed by four appendices:

Part I, The Unify DataServer/ELS Environment, contains six chapters. These chapters explain the environment that Unify DataServer/ELS operates in.

Chapter 1 Introduction to Unify DataServer/ELS is an introduction to the Unify DataServer/ELS Relational Database Management System.

Chapter 2 Unify DataServer/ELS and the Operating System Environment explains how Unify DataServer/ELS works on your computer'& operating system.

Chapter 3 The Menu Handler discusses the Unify DataServer/ELS user-friendly, menu-driven system. The Menu Handler has all the options you need for application development and operations.

Chapter 4 Files and Directories explains the kinds of files you work with in Unify DataServer/ELS and the file-naming conventions you use to identify them. It also outlines the directories these Unify DataServer/ELS files reside in: system directories and application or user directories.

Chapter 5 Environment Variables explains how to specify parameters to use with Unify DataServer/ELS and your database application. Environment Variables let you set such things as path names, date display format, and a text editor or word processor.

Chapter 6 Termcap and Unicap has information to help you set up your terminal and keyboard to work with Unify DataServer/ELS.

Part II, Database Design and Maintenance, contains three chapters.

Chapter 7 Introduction to Database Design contains information that can help you set up the most efficient database design for data entry, manipulation, and retrieval.

Chapter 8 Database Design Utilities tells you how to, design and create new databases, modify existing databases, reconfigure databases, work with B-tree indexes, and use Advanced Field Attributes for data validation.

Chapter 9 Database Maintenance Utilities explains how to read and write database backups, rebuild, the hash table, rebuild explicit relationships, and define database volumes.

Part III, System Administration, also contains three chapters.

Chapter 10 Database Security Utilities tells you how to manage security for your Unify DataServer/ELS application. You can. define group and user privileges for database access and operations, and set up field passwords. If your operating system supports this feature, you can also

combine the Unify DataServer/ELS security system with that of your operating system.

Chapter 11 Data Dictionary Reports explains several reports you can request to see the status of your database, menus, screens, security, programs, and other statistics relating to your application.

Chapter 12 Miscellaneous Utilities describes such operations as how to Add or Delete Menus, Describe Programs to the System, Edit SQL or RPT Scripts, use DBLOAD, and do Transaction Logging.

Transaction Logging records database updates so that if the system crashes, you can restore all changes made since the last checkpoint.

Part IV, Screen Form Development and Maintenance, contains two chapters.

Chapter 13 Screen Form Development Utilities and Screen

Chapter 14 Form Maintenance Utilities have information to help you use screen forms. These chapters tell you how to create default screen forms for data entry and database inquiries, then customize those screen forms for your application.

These chapters also discuss the tools you can use to maintain the forms after you've created and customized them.

Part V, Data Entry and Retrieval, also contains two chapters.

Chapter 15 ENTER-Data Entry/Query By Forms, describes ENTER. This is a general-purpose data entry program that you can use to drive screen forms. You can also use ENTER as a query-by-forms interface to your database or as the basis for your own custom data-entry screens.

Chapter 16 SQL-Query/DML Language explains how to use the Unify DataServer/ELS query language, SQL, to extract information from your database in a meaningful format.

SQL is a subset of the IBM standard relational query and data manipulation language, SEQUEL 2.

Part VI, Report Generators, contains the last two chapters.

Chapter 17 RPT-Report Processor tells you how to use the Unify DataServer/ELS Report Writer, RPT, to print reports. With RPT, you use a non-procedural language to specify the format of multiple-level, tabular reports. The data for these reports can come from an SQL or Query-By-Forms query, a user program, or any ASCII file.

Chapter 18 LST-Listing Processor explains LST, a selection and formatting language that produces sorted listings of totals and subtotals from a database. LST is a simple alternative to SQL and RPT, when you don't need the full power of those two tools.

LST has two major components, the selection processor, which lets you choose records based on selection criteria, and the listing processor, which lets you sort, format, and total the selected records.

Appendices A through D contain supplementary information.

Appendix A Field and Internal Data Types explains field data types, display lengths, and internal lengths, as well as legal changes you can make to field types and lengths.

Appendix B Reserved Words contains a list of words that have specific meaning to Unify DataServer/ELS. You cannot use these reserved words for table and field names.

Appendix C Customizing Your Application describes ways to tailor your Unify DataServer/ELS database application to your needs. Custom Message Control and a Reconfigurable Data Dictionary enable you to personalize screen messages and prompts. You can also set the CURR, DATETP, and LDATEFMT environment variables for currency and date display formats.

Appendix D Error Messages contains the messages that can display while you are running Unify DataServer/ELS.

Syntax Conventions

Throughout the manual, standard conventions are followed to describe the syntax of various Unify DataServer/ELS and shell commands:

[ ]	Words within square brackets are optional, and may be left out.
| |	Vertical bars enclosing a stacked list of options means to choose one of the options, and only one.
...	Ellipsis points indicate that the immediately preceding item may be repeated any number of times.

Syntax conventions are also outlined in chapter 16, SQL-Query/DML Language, and chapter 18, LST-Listing Processor.

Other Conventions

Warnings indicate actions that could cause data loss or damage to the database.

NOTE:	Notes indicate conditions of interest or actions that could result in syntax errors but not damage the database or cause loss of data.

Part I: The Unify DataServer/ELS environment

Part I explains the Unify DataServer/ELS operating environment. The chapters in Part I describe Unify DataServer/ELS menus, files, general directory layout, shell environment variables, and the termcap and unicap files. Although the use of Unify DataServer/ELS requires little or no knowledge of the computer's operating system, the shell, or programming, the information in these chapters is a little more demanding. To understand this information, you should know something about programming principles.

You should be familiar with directories, the shell, and, for the chapter on termcap and unicap, be familiar with how CRT terminals operate from a programming viewpoint. The chapters in Part I contain information about miscellaneous items not discussed elsewhere in the manual. However, this information may help make your Unify DataServer/ELS run smoothly.

Because Part I addresses most of the system in some way, it includes many references to other sections of the manual. If you are not yet too familiar with Unify DataServer/ELS, don't worry about whether you completely understand this part. It is still a good idea to read Part I as an orientation and then to use the system. When you are more familiar with Unify DataServer/ELS, review this part, and its information will be much clearer.

Chapter 1: Introduction to Unify DataServer/ELS

This chapter introduces the Unify DataServer/ELS Relational Database Management System. Any database management system is more than a single program, unlike word processors or spread sheet packages that generally are just single programs.

Unify DataServer/ELS is a collection of over 20 different programs, integrated to let you create and modify application systems that store and retrieve data. Figure 1-1 shows a diagram of the Unify DataServer/ELS system architecture.

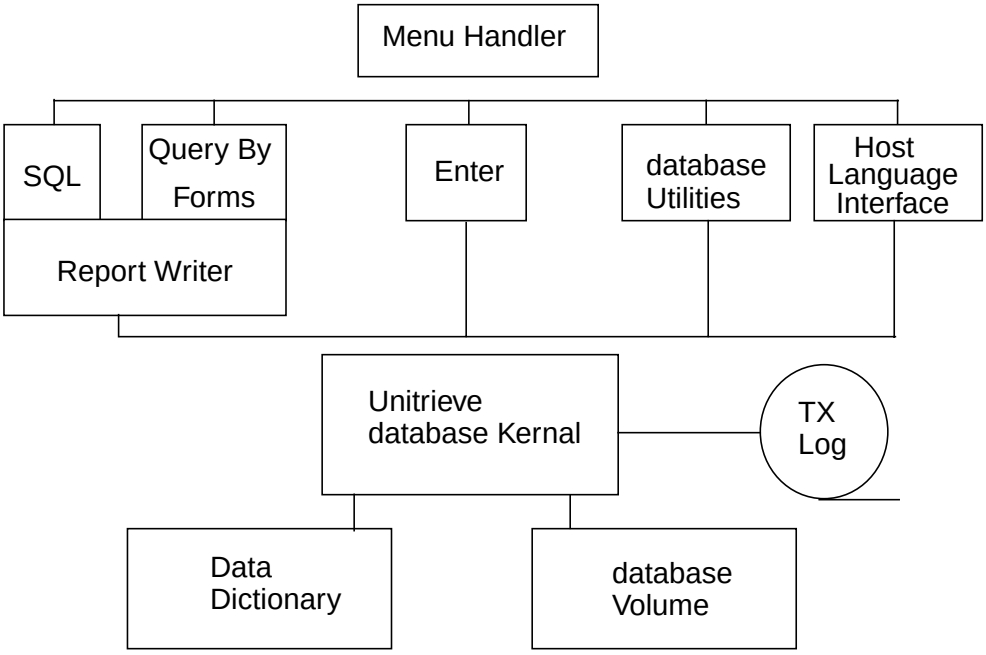


Figure 1.1 Unify DataServer/ELS system architecture

The primary user interface to the Unify DataServer/ELS system is the Menu Handler. The system administrator can select any Unify DataServer/ELS program from a

menu, while also controlling security to permit other users access only to certain menus and programs.

Unify DataServer/ELS comes with a set of built-in menus ready for you to use in developing your application. See chapter 3, The Menu Handler, for a complete description of how to use menus. You can also create your own menus. For more information, see Add, Modify or Delete Menus, section 12.1.

Before you design your own database application, you should review the concepts of database design and access methods, explained in Introduction to database Design, chapter 7.

Chapter 9, Database Design Utilities, and chapter 10, database Maintenance Utilities describe the options to create and maintain a database. These include programs to enter and modify the database design, create or reconfigure the database, add or drop indexes, read and write backup tapes (or removable disks), and repair corrupted databases. You can also specify that a Unify DataServer/ELS database is to reside on a *raw file system*. This yields substantial performance improvements over using an ordinary ASCII file.

Full-screen data entry is provided by ENTER as described in chapter 15, ENTER-Data Entry/Query by Forms. ENTER performs many edit checks, including type, length, date and time formats, dollar amount formats, and table lookup. You can also add your own functions to customize data entry. As with the previous programs, ENTER uses the screen forms that you create and modify.

Using SQL, you can interactively add, modify, delete, and query information in your database. The features of SQL are described in chapter 16, SQL Query/DML Language.

Unify DataServer/ELS also includes some additions and extensions to SQL worth noting. For example, you can connect a screen form to an SQL query script, then format the results of the query with the primary Unify DataServer/ELS report writer, RPT. This interface, called *SQL By Forms,* is described in SQL/Screen Form/Report Interface, section 16.4.

For simpler queries, often more than half of the queries performed, Unify DataServer/ELS offers *Query By Forms*. With QBF, you first fill in search values on a screen form, and Unify DataServer/ELS then uses these values to find the records that match.

You can look at the selected records one at a time on the form, or you can send the results to either of the Unify DataServer/ELS report writers: RPT (powerful) or LST (easy to use). QBF is described in ENTER Data Entry/Query By Forms, chapter 15; RPT in chapter 17; LST (commands for formatting reports) in chapter 18; and the way to combine these functions, in Register Screen Form with ENTER, chapter 15.1.

Building a Unify DataServer/ELS database application

Building a Unify DataServer/ELS database application requires seven basic steps. The following paragraphs briefly describe each step and how you can find the information you need to complete it:

1. Develop a logical database design.

   To develop your database's logical design, you must analyze your database application requirements. You need to determine the types of information the database must store and how it should organize that information.

2. Install the Unify DataServer/ELS software.

   Instructions for installing the database software are given in the Unify DataServer/ELS Installation Guide.

3. Set up the database environment.

   To set up the environment for your database application, you configure the operating system environment so your application runs as efficiently and smoothly as possible. For information about the database and operating system environments, see chapters 1 through 5 of this manual.

4. Create the database.

   You create the database by defining the database tables and the fields for each table. For information about, creating, a new database, see chapter 8, "Database Design Utilities."

5. Enter or load data into database tables.

   To enter or load data in database tables, you can use either of two methods:

   - use the database Load (DBLOAD) program to load records from an ordinary ASCII file
   - use an ENTER screen for data entry

For more information about how to enter data, see Section 12.6, "Database Load" or Section 15.2, "Using ENTER for Data Entry."

6. Create screens, scripts, or reports as needed.

   You will want to provide a way for users to view data. Depending on the type of information needed, use one or more of the following methods for displaying data:

   - Create screen forms for viewing or querying records interactively from the screen. For detailed information, see chapter 13, "Screen Form Development Tools."
   - Create SQL scripts for querying the database and printing simple lists of data. For detailed information, see chapter 16, "SQLQuery/DML Language. "
   - Create RPT scripts for running routine reports with customized formats. For detailed information, see chapter 17, "RPT Report Processor."

7. Set up security and add users to the database.

   To set up database security and add users to the database, you must determine how much of the database you want users to be able to access and which operations you want to allow these users to perform.

   For general information about database security, see chapter 10, "database Security Utilities."

If you have never used Unify DataServer/ELS before, start by reviewing the *Unify DataServer/ELS Developer's Tutorial Manual*. It gives examples and step-by-step instructions for using Unify DataServer/ELS tools to develop an application system.

References

For more information on designing and managing an RDBMS, or on managing transactions, locking, and recovery, the following sources are suggested:

Bernstein, Philip A., Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Reading, Mass.: Addison-Wesley, 1987.

Chamberlin, Donald D. "Relational Data-Base Management Systems." *Computing Surveys 8* (March 1976).

Date, C.J. *An Introduction to Database Systems*, 3rd ed. Reading, Mass.: Addison-Wesley, 1982.

Martin, James. *Computer Data-Base Organization*, 2nd ed. Englewood Cliffs, N.J.: Prentice-Hall, 1975.

Salzberg, Betty. *An Introduction to database Design*. Academic Press, 1986.

Salzberg, Betty. "Third Normal Form Made Easy." *SIGMOD Record* 15, No. 4 (December 1986).

Chapter 2: Unify DataServer/ELS and the operating system environment

In developing your Unify DataServer/ELS application, you sometimes have to deal directly with your computer's operating system. Consequently, the Unify DataServer/ELS Developer's Reference Manual often refers to shell commands or operating system commands.

The manual assumes that you already know how to use your operating system, but this chapter is included as a summary of how Unify DataServer/ELS interfaces with Unix-based operating systems.

The following table lists some of the basic operating system commands you are expected to be familiar with.

Command	Description
cat	display contents of an ASCII file
cd, chdir	change directory
cmp	compare two files
cp	copy a file
df, fsck	check disk
ls	list files and subdirectories in a directory
mkdir	create a new directory
mv	change a file name
pr, lpr	send output to a print spooler
rm	delete a file
rmdir	delete a directory
setenv, export	set an environment variable
vi	open the standatd text editor

Figure 2.1 Summary of Operating System Commands

File name suffixes

Unify DataServer/ELS identifies the type of a file by its file name suffix. For more information on file names and suffixes, see Files. This table lists the file name suffixes recognized by Unify DataServer/ELS.

Suffix	Description
.a	an archive file created by ar
.afa	an advanced field attribute file
.c	a C source code file
.db	a data base file
.dbr	a raw data base file
.dbv	a file containing variable-length data base fields (TEXT or BINARY fields)
.err	an error log file
.h	an include file used with C source code
.idx	a B-Tree index file
.id	a command file used to create executable files
.o	a relocatable binary object file usually kept in an archive file
.q	a Unify DataServer/ELS screen form file read at run time

Figure 2.2 File name suffixes recognized by Unify DataServer/ELS

Directories and path names

Unify DataServer/ELS has several environment variables you can set to specify directories and path names. The list on the following page introduces these environment variables. For more information about Unify DataServer/ELS environment variables, see chapter 5.

Before setting environment variables, you should know which environment you are working on; for example. Bourne Shell or C Shell. You should also be familiar with environment variable settings in the .login or .profile file.

If you need more information about environment variables on your operating system, refer to your operating system user's manual or one of the source books listed at the end of this chapter.

- DBPATH contains the complete path name for application database files. If DBPATH is not set, all application database files are assumed to be in the current directory.
- LOCKDIR contains the name of the named pipes directory. If LOCK.DIR is not set, the default named pipes directory is /tmp.
- PATH contains the name of the directory where Unify DataServer/ELS system programs are located.
- TERM CAP contains the complete path name of the termcap file. The default location of the termcap file is in the /etc/termcap directory.
- UNICAP contains the complete path name of the unicap file. The default location of the unicap file is in the directory specified in Unify DataServer/ELS.
- Unify DataServer/ELS contains the name of the Unify DataServer/ELS system lib directory,
- UNIFYTMP contains the name of the temporary file directory used by Unify DataServer/ELS. The default value of UNIFYTMP is /tmp.
- VOLPATH contains the complete path name for database volumes. If VOLPATH is not set, all database volumes are assumed to be in the current directory or in the directory specified in DBPATH.

Text editors

The Unify DataServer/ELS default text editor is vi, the Berkeley visual editor. However, if you have a text editor you would rather use, you can specify that text editor by setting the EDIT environment variable.

You can also specify a binary editor by setting the BINEDIT environment variable.

The print spooler

The SPOOLER environment variable contains the name of your system's print spooler. This variable is set to the standard spooler, lpr. (The lpr is a leftover from the days when all high speed printers were line printers.)

When Unify DataServer/ELS sends output to a printer, the print spooler makes a temporary copy of the file. The temporary spooler file is removed when printing is finished.

Running operating system commands from Unify DataServer/ELS

You can run operating system commands from Unify DataServer/ELS in two ways:

- You can access the operating system shell with the Unify DataServer/ELS system sh command.
- You can run operating system commands from SQL using the ! command.

To access the operating system shell from Unify DataServer/ELS, enter sh at the selection prompt as follows:

```
SELECTION: sh
ESC-select  ^U-up  RET-down  ^X-home  ^P-previous  ^Z-clear  ^D-exit
```

A screen similar to the following one displays:

```
[sh]                    Unify RDBMS          30 APR 1999
                  Operating System Shell

unify >
```

At the shell prompt, enter the operating system commands you want to run. To return to the Unify DataServer/ELS menu, press CTRL D.

To run operating system commands from SQL, enter ! followed by the operating system command. For example, to see a list of the files in the current directory, enter ! and the Is command at the SQL prompt, as shown:

```
sql> !ls
ddlockfile   file.db    unify.db
sql>
```

The command runs, displays its results, and returns to the SQL prompt.

Using operating system security from Unify DataServer/ELS

Unify DataServer/ELS provides a security maintenance system that you can use with your operating system to prevent unauthorized access to your database application. Unify DataServer/ELS lets you specify a super-user ID and password, group and individual access privileges, and database field passwords.

In addition to the Unify DataServer/ELS security measures, you can specify an operating system ID and password for your application's system administrator, and make sure all system files are owned by this ID.

For more information about Unify DataServer/ELS system security and how to use it with your operating system security, see chapter 10.

If you need more information about your operating system, refer to your operating system user's manual or one of the following sources:

Bell Telephone Laboratories, Inc. *UNIX Programmer's Manual.* New York: Holt, Rinehart and Winston, 1983.

McOilton, Henry, and Rachel Morgan. *Introducing the UNIX System*. New York: McOraw-Hill, 1983.

Thomas, Rebecca, and Jean Yates. *A User Guide to the UNIX System*. New York: Osbome McOraw-Hill, 1985.

Chapter 3: The Menu Handler

The Menu Handler provides a complete, user-friendly environment for the Unify DataServer/ELS system. You use menus to develop a project, to create an application, and to operate the finished application. All application development and operations are menu-driven.

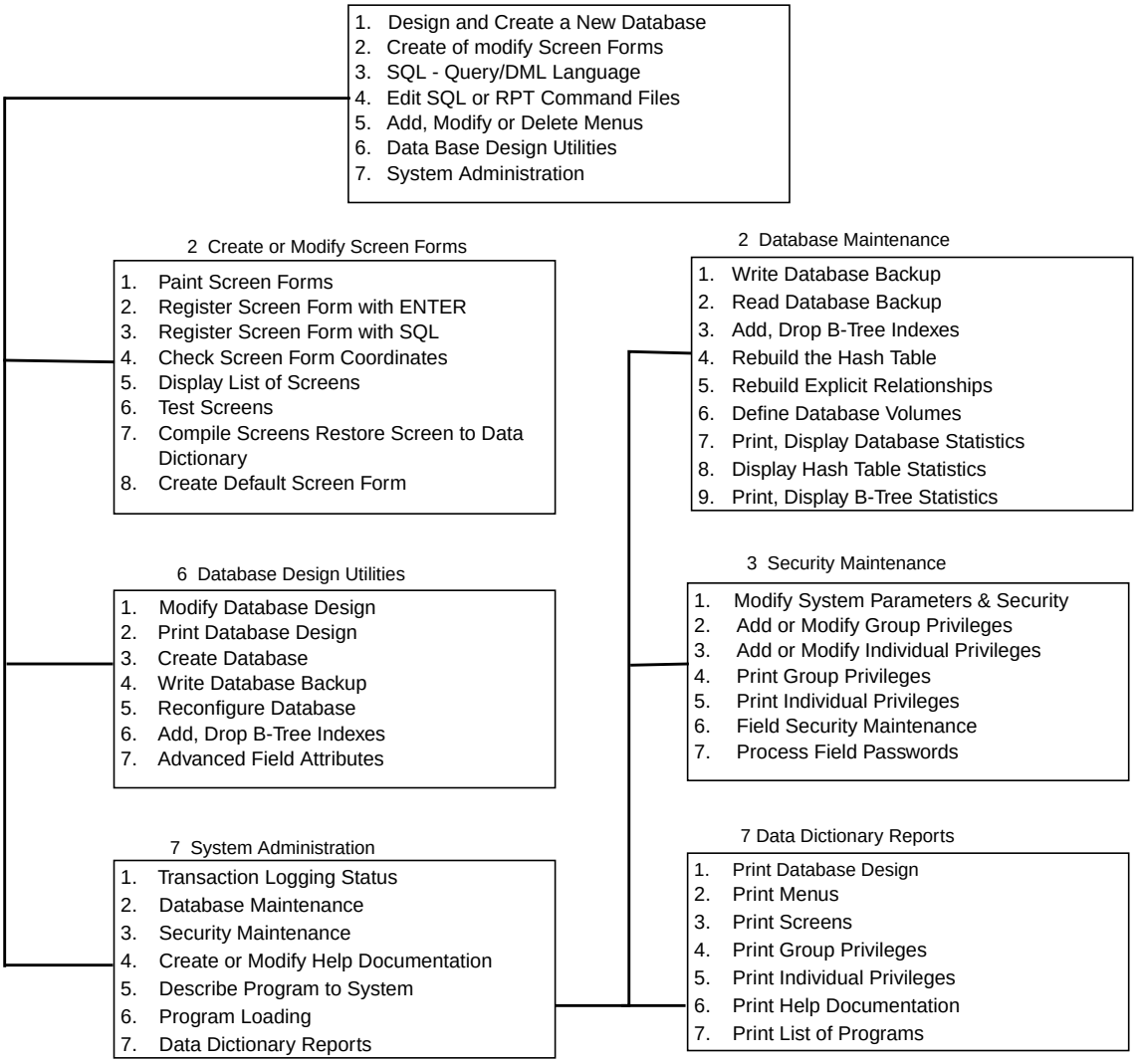
Several different Unify DataServer/ELS options let you update the data dictionary to keep track of the menus, application functions, ENTER screens, and users of an application system. You can also store information about where to find the archives for each application program, what programs are loaded together in a single executable core load, and what screen form is to be displayed when a program begins execution.

Standard Unify DataServer/ELS menus

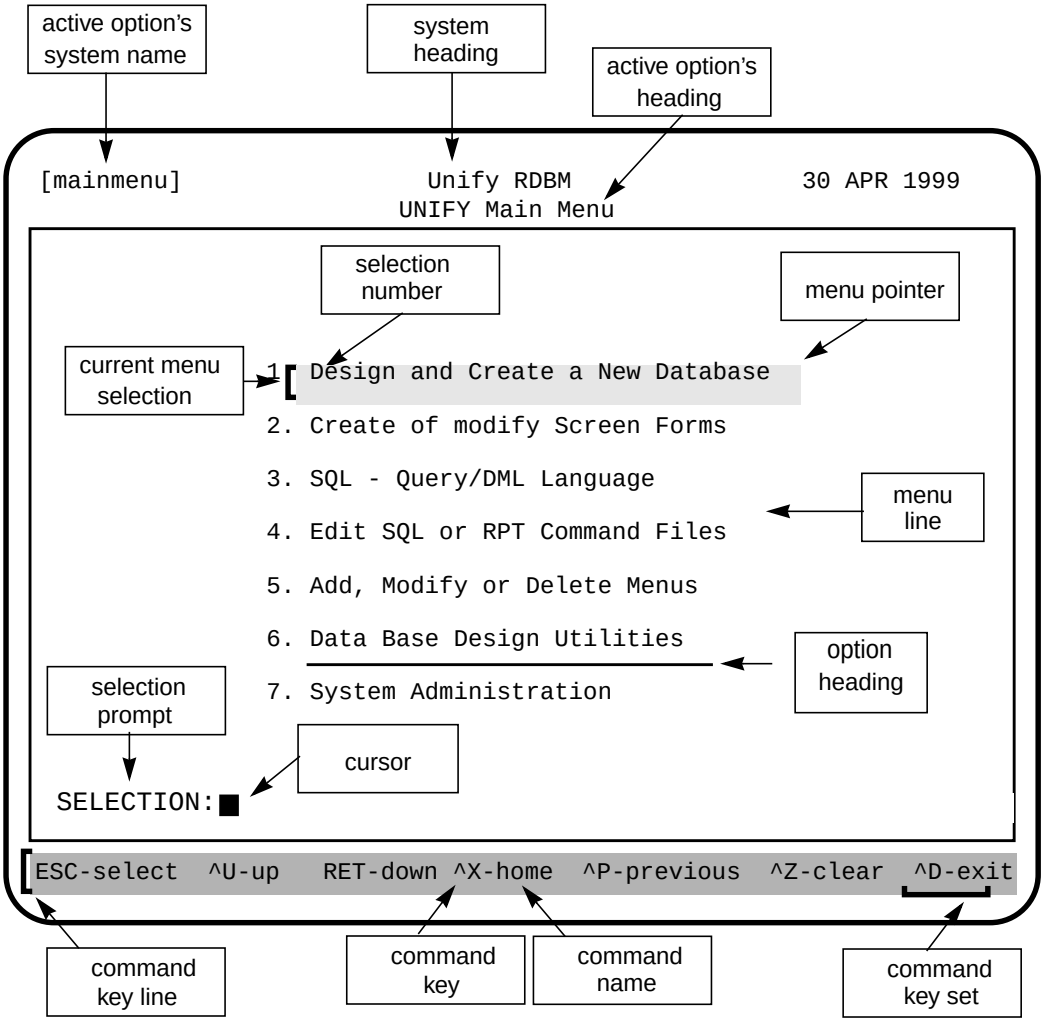
A *menu* is a set of Unify DataServer/ELS options, displayed for selection. Your initial empty data dictionary contains only the Unify DataServer/ELS programs, menus, and a user ID. You can use the Add, Modify or Delete Menus program to create custom menus for the application and arrange the menus in a hierarchical menu system.

On the next page is a diagram of the standard menus that come with Unify DataServer/ELS.

Menu Map


1. Design and Create a New Database
2. Create of modify Screen Forms
3. SQL - Query/DML Language
4. Edit SQL or RPT Command Files
5. Add, Modify or Delete Menus
6. Data Base Design Utilities
7. System Administration
2 Create or Modify Screen Forms
1. Paint Screen Forms
2. Register Screen Form with ENTER
3. Register Screen Form with SQL
4. Check Screen Form Coordinates
5. Display List of Screens
6. Test Screens
7. Compile Screens Restore Screen to Data Dictionary
8. Create Default Screen Form
6 Database Design Utilities
1. Modify Database Design
2. Print Database Design
3. Create Database
4. Write Database Backup
5. Reconfigure Database
6. Add, Drop B-Tree Indexes
7. Advanced Field Attributes
7 System Administration
1. Transaction Logging Status
2. Database Maintenance
3. Security Maintenance
4. Create or Modify Help Documentation
5. Describe Program to System
6. Program Loading
7. Data Dictionary Reports
2 Database Maintenance
1. Write Database Backup
2. Read Database Backup
3. Add, Drop B-Tree Indexes
4. Rebuild the Hash Table
5. Rebuild Explicit Relationships
6. Define Database Volumes
7. Print, Display Database Statistics
8. Display Hash Table Statistics
9. Print, Display B-Tree Statistics
3 Security Maintenance
1. Modify System Parameters & Security
2. Add or Modify Group Privileges
3. Add or Modify Individual Privileges
4. Print Group Privileges
5. Print Individual Privileges
6. Field Security Maintenance
7. Process Field Passwords
7 Data Dictionary Reports
1. Print Database Design
2. Print Menus
3. Print Screens
4. Print Group Privileges
5. Print Individual Privileges
6. Print Help Documentation
7. Print List of Programs

When you start Unify DataServer/ELS, this is the first menu that displays:



All Unify DataServer/ELS menus are similar to this one. The top two lines display the active menu's system name (mainmenu), descriptive heading (Unify DataServer/ELS Main Menu), and the current date and time. (These top two lines are also the same for all programs and screen forms.)

Centered on the menu is a list of the options that you can select using this menu. These options are listed as *menu lines*. A menu line consists of the option's descriptive heading preceded by a line number. This number is also known as the *selection number*. Unify DataServer/ELS allows up to 16 menu lines per menu.

The *current menu selection* is marked by the *menu pointer*. The menu pointer is represented on the screen by either reverse video text or an asterisk. The type of menu pointer used on your screen depends on your terminal's capabilities and its termcap entries (see Termcap).

Near the bottom of the screen is the *selection prompt* — SELECTION:. You enter commands or selection numbers here to indicate your choice.

At the bottom of the menu screen is the Menu Handler *command key line*. This line displays the Menu Handler commands. Each command is represented as a pair of items—*the command key* and the actual *command name*. This pair is known as a *command key set*.

The command key is a keystroke that executes a Menu Handler operation. The command name gives you an idea of what will happen when you press the command key. Each command key set is displayed in the following format:

```
command key-command name
```

COMMAND NAMEACTION	NAME	COMMAND KEY
select highlighted option	select	ESC F1
move menu pointer up	up	^U (UP ARROW) ^K
move menu pointer down	down	RETURN (DOWN ARROW) ^J LINE FEED
go to home menu	home	^X HOME F2
go to pervious menu	previous	^P F3

Figure 3.1 Unify DataServer/ELS command key default settings

COMMAND NAMEACTION	NAME	COMMAND KEY
clear selection line	clear	^Z DELETE F4
exit Unify DataServer/ELS	exit	^D F5
display general Menu Handler help	help	? F6
display more command keys	more	/ F10
display information about the highlighted menu line	info	TAB F6 ^I
redraw screen	redraw	^R F8
toggle command key line on/off	toggle	^T F9

Figure 3.1 Unify DataServer/ELS command key default settings

In the previous table, the ^ symbol represents the CTRL key: ^X means CTRL X. You produce CTRL X by holding down the CTRL key and pressing the X key. In other words, you use the CTRL key just like the SHIFT key. You can customize the command key definitions by modifying your unicap file. (See section 3.1, Customizing Menu Handler Cammands.)

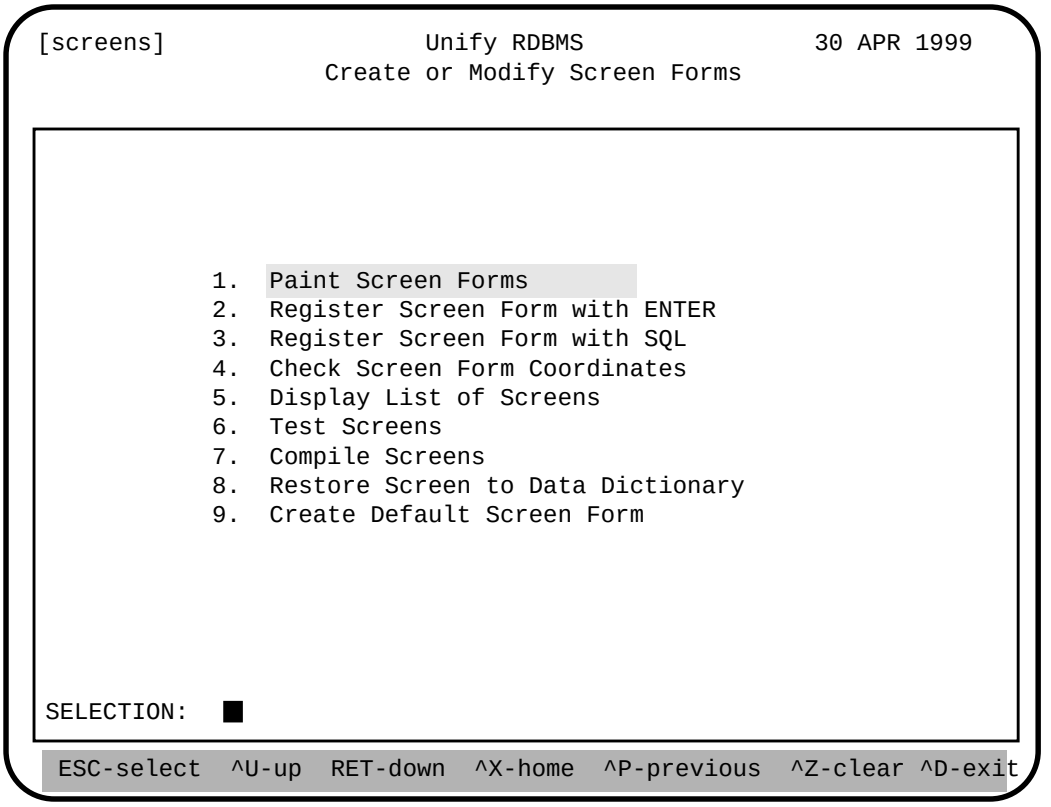
Looking at the Main Menu, you initially have seven options to choose from. You can make a selection in one of three ways: by selection number, by select, or by system name. The following explains how to use each selection method:

1. Enter a selection number and press RETURN to run the associated option.
2. Press the *select* command key to run the option highlighted by the menu pointer (the current menu selection).
3. Enter the system name for a program, option, or menu. Then press RETURN to run that program, option, or menu. These system names are assigned by Unify DataServer/ELS or by you when you create menus, ENTER or SQL screens, or custom programs.

Registered options can be run from any menu by entering the system name at the selection prompt. For example, at the Unify DataServer/ELS Main Menu, you can go

directly to Print Help Documentation by typing prthdoc. This is a major time saving feature for experienced users.

To experiment with these selection methods, start by entering screens at the selection prompt. The Create or Modify Screen Forms menu displays as follows:



At the Create or Modify Screen Forms menu, you can move to the fifth menu line with the *down key*. Use the *select* command key to select Display List of Screens. When Display List of Screens is finished, the Create or Modify Screen Forms menu redisplays.

To return to a previous menu, press the *previous* command key. If you are at your home menu, an error message displays at the bottom of the screen, to tell you there is no previous menu. If you are at the Create or Modify Screen Forms menu, the *previous* command key causes the Unify DataServer/ELS Main Menu to redisplay. Notice that the menu pointer highlights the last selection you made.

Unify DataServer/ELS users see only the menus and programs that match their security privileges. These security privileges are set by the system administrator using Add or Modify Group Privileges and Add or Modify Individual Privileges. The Menu Handler also prevents users from accessing programs or menus by name for which they have no privilege. chapter 10, Database Security Utilities describes how this works.

Users

Entry to the Unify DataServer/ELS system can be controlled with one of two different login modes. The first mode allows access to the database without a login screen; this is the default mode. This method does not enforce Unify DataServer/ELS security. Users have access to all programs, menus, and options. The second mode uses a login screen that requires the user to enter a user ID and a password. The display of the login screen can be set by the system administrator using Modify System Parameters & Security.

Using the login information, Unify DataServer/ELS determines which menu users see when they start Unify DataServer/ELS (the *entry menu*, also known as the *home menu*), and which options they are allowed throughout the application. In addition, when a program is run, Unify DataServer/ELS passes the user's current access privileges to that program.

Each user has a defined access privilege to every program and menu. This ranges from no access privilege (the default condition) to privileges enabling the user to inquire, modify, delete, or add. The four access capabilities can be used in any combination. For example, a user may have the right to inquire about and delete records in a database, to modify and add records, or to inquire about records only. Or the user may have all four access privileges.

In addition to access privileges, the login ID sets an *entry menu* for a user. This menu is the first one seen after logging in to Unify DataServer/ELS, and is the one returned to when the user presses the *home* command key. This means a user may only see that portion of the database menu tree that applies to that user. Because programs can appear on multiple menus, the system administrator can tailor each user's environment exactly as required.

As a convenience, each user may be assigned to a *user group*. This group is a collection of users with similar access privileges. You can assign a default set of privileges to the group and then set individual differences for each user if necessary.

For example, data entry clerks may be entered as a group because, for the most part, they need access to the same programs Likewise, warehouse workers may need their own set of programs. A new employee, however, may temporarily need some of the more sensitive programs restricted. In addition, the new employee may only have inquire privileges for some programs. A supervisor may have access to more programs and be given the addition of delete privileges.

Finally, there is a special *super user* in the Unify DataServer/ELS system. This user automatically has access to all options. The super user ID, password, and entry point are all entered with Modify System Parameters & Security. By default, the ID and password are both su and the entry menu is the Unify DataServer/ELS Main Menu (mainmenu).

Help documentation

Unify DataServer/ELS allows help documentation to be associated with every program, screen, and menu. Your release of Unify DataServer/ELS has help documentation for all options and SQL commands. Help text for custom programs and screens can be entered using Create or Modify Help Documentation.

You can display help documentation from a menu in the following ways:

1. Press the *help* command key, ?.

   Displays general information about how to use the Menu Handler.

2. Press the *info* command key, CTRL I.

   Displays the help documentation for the current menu selection.

3. Enter help at the selection prompt.

   Displays help documentation for the active menu.

4. Enter help n at the selection prompt.

   Displays the help documentation for the option with selection number n.

5. Enter help xxxx at the selection prompt.

Displays help for the option whose system name is xxxx.

Executables

An executable is a shell script or a compiled program that you can run as a command from the operating system shell. The shell is the operating system level where you enter the command to start Unify DataServer/ELS. Custom programs, or executable, can be registered with Unify DataServer/ELS. When registering an executable with Unify DataServer/ELS, you must specify the heading that is to appear on every menu line where this program appears. This heading is also placed at the top of the screen when the Menu Handler executes the program. You can also specify a screen form which should be displayed before the program begins to run, and the source directory where the program's object code archive is located. The latter information is used by Program Loading (see the Unify DataServer/ELS Direct HLI Programmer's Manual).

For detailed information on registering executables with Unify DataServer/ELS, see Describe Program to System, section 12.3.

Starting Unify DataServer/ELS automatically

The system administrator can set up the Unify DataServer/ELS startup program, unify, as the shell. When users log into the operating system, they will see the Unify DataServer/ELS login screen instead of the standard shell. Or, the system administrator can arrange to bypass the Unify DataServer/ELS login screen entirely and have users see their first Unify DataServer/ELS menu upon logging into the operating system.

The startup file for the Menu Handler is named unify. This is the command that you enter to display the Unify DataServer/ELS login screen and begin to run a Unify DataServer/ELS application. The unify file also sets the appropriate environment variables, exports them, and starts the Unify DataServer/ELS shell script. Unify DataServer/ELS is in the Unify DataServer/ELS bin directory and actually displays the menus and controls security.

To define unify as the shell for a user, first set the appropriate entry in the operating system password file. Then set up the .login or .profile file to establish the correct environment, and run unify.

If you do not use the DBPATH environment variable, the user home directory must be the application bin directory (see Directories, section 4.2).

Customizing Menu Handler commands

In the standard version of the Menu Handler, each command is invoked by using the keystroke (command key) described in the unicap file. The key emits a character (or character sequence) that the Menu Handler recognizes as being associated with a particular command. You can change any command definition to use a different command key by editing the menu section of the unicap file. For more information about unicap, see section 6.1.

The list of internal command names used by the Menu Handler in the unicap file is shown in Figure 3.2, following:

Name	Action
ignoremiss	report missiong command definitions
more	display more command key sets at the bottom of the screen
select	run the highlight menu line
up	move menu pointer up
down	move menu pointer up
home	go to home menu
previous	go to previous menu
clear	clear selection line
exit	exit UNIFY
help	display general Menu Handler help
info	display information about the highlighted menu line
redraw	redraw screen
toggle	toggle command key line on/off

Figure 3.2 UNIFY Menu Handler Commands

Figure 3.3 is an example of the standard Menu Handler section in the unicap file:

```
SECTION = menu          menu handler
ignoremiss = FALSE      report missing command definitions
more = /  | <f10>
select= \E | <f1>
up = ^u | <up_arrow> | ^k
down = \r | ^j | \n | <down_arrow>
home = ^x | <home> | <f2>
previous = ^p | <f3>
clear = ^z | \177 | <f4>
exit = ^d | <f5>
help = ? | <f6>
info = ^i | <f7>
redraw = ^r |<f8>
toggle = ^t | <f9>
```

Figure 3.3 Menu Handler Section in unicap File

All the command names can be associated with command keys of your choice. As an example, consider the following line in the unicap file that defines the command key to exit Unify DataServer/ELS:

```
exit, = ^d
```

The command name appears on the left, exit, while the command key appears on the right, ^d - CTRL D. If you want to exit when you press CTRL W instead: you change ^d to ^w.

Chapter 4: Files and directories

All database information and programs used in Unify DataServer/ELS are stored in files. Unify DataServer/ELS identifies files by their file names. File names usually have specific suffixes that tell Unify DataServer/ELS what kind of information the files contain. For example, a .q suffix tells Unify DataServer/ELS that the file is a screen form file. File names and suffixes are explained in below in "Files".

To use files, Unify DataServer/ELS not only needs to know what kind of information the files contain, but also where the files are located. Unify DataServer/ELS usually expects to find files in specific directories. For example, it looks for database help documentation in the hdoc directory below the user's root directory. "Directories", explains the Unify DataServer/ELS directory system.

4.1 Files

Unify DataServer/ELS uses file naming conventions to identify the different kinds of files necessary to operate an application. File names have the form *name .suffix*, where both the *name* and the *suffix* are variable length. The suffix indicates the type of file, as with the usual operating system file naming conventions.

The following is a list of Unify DataServer/ELS file name suffixes, their meanings, and a description of significant files with that suffix. Unify DataServer/ELS files are usually located in a particular directory, which is noted here. The Directories section describes the directory layout in more detail.

The following are the file name suffixes recognized by Unify DataServer/ELS:

.a	An archive file, created with the operating system command ar, that contains relocatable binary files in a form suitable for loading using the Id command. Unify DataServer/ELS has some system archives that are in the system lib directory. The application developer can also create archive files, usually in a src subdirectory.
.afa	An Advanced Field Attributes file. Unify DataServer/ELS uses two attribute files, unify.afa and fields.afa. The file fields, afa is the Advanced Field Attributes ASCII source file. The file unify.afa is the compiled version of fields.afa. The Advanced Fields Attributes option is explained in the section Advanced Field Attributes.
.b	A C source code file. This kind of file is created by the application developer, and is typically located in an src subdirectory.
.db	A database file. Every application has two database files—the data dictionary, which is called unify, db, and the application database file, called file.db. The user's unify, db is copied from the system prototype unify, db the first time Unify DataServer/ELS is started, and file. db is created with Create Database (section 8.3). The user's database files are located in the database bin directory. The prototype data dictionary is located in the system lib directory.

.dbr — The raw database file, .named file. dbr. It is created by Create Database (section 8.3) if you have used Define Database Volumes and have defined the root volume as a device volume. The raw database file, file. dbr, is linked to the character device, file.db is linked to the block device.

.dbv — A file containing variable-length database fields. TEXT or BINARY type fields are stored in .dbv files.

.err — An error log file. The most common file with this suffix is afa. err. This is the log for errors detected while using Advanced Field Attributes. Two other error log files are repoint.err and erriog.

.h — An include file, used with C source code. Unify DataServer/ELS creates two kinds of .h files: file. h and screen, h. The file.h, file is the result of a Create Database (section 8.3) or Reconfigure Database (section 8.4). This file contains a list of table and field names used by C programs that access the database.

The screen.b files are created by either Paint Screen Forms (section 13.2) or Compile Screens (section 14.4). These files contain a list of screen field names used for that screen form by C programs. Although created in the bin directory, the .h files should be moved to either the def or include directories.

.idx — A B-tree index file. These are binary files that reside in the database bin directory. They are created by Add, Drop B-Tree Indexes (section 8.5), and are updated by any program that stores data into an indexed field. The files named bt/mn.idx (nnn is a number) are actual B-tree files. The file field, idx is a directory to the index files.

.ld — A shell command file that uses the operating system command ld to create executable files. These usually reside in the user's build directory.

.o — A relocatable binary file that is the result of a cc-c. These files are usually kept in an archive file.

.q — A Unify DataServer/ELS screen form file. These files contain binary versions of screen forms which can be read quickly at run time. They are produced by either Paint Screen Forms (section 13.2) or Compile

Screens (section 14.4). Unify DataServer/ELS system screen forms are kept in the system lib directory. A user's screen forms are usually stored in the database bin directory.

4.2 Directories

Unify DataServer/ELS uses two types of directories: Unify DataServer/ELS system directories; and application directories.

Unify DataServer/ELS system directories are used by all Unify DataServer/ELS users and are the ones delivered with your Unify DataServer/ELS release. The contents of Unify DataServer/ELS system directories should not be modified.

Each application has its own set of application directories. These contain the files specific to a particular application.

You can set environment variables to specify the directory locations for Unify DataServer/ELS and application files. This is described in chapter 5, Environment Variables.

The diagram in Figure 4.1 shows the usual structure of the Unify DataServer/ELS directories:

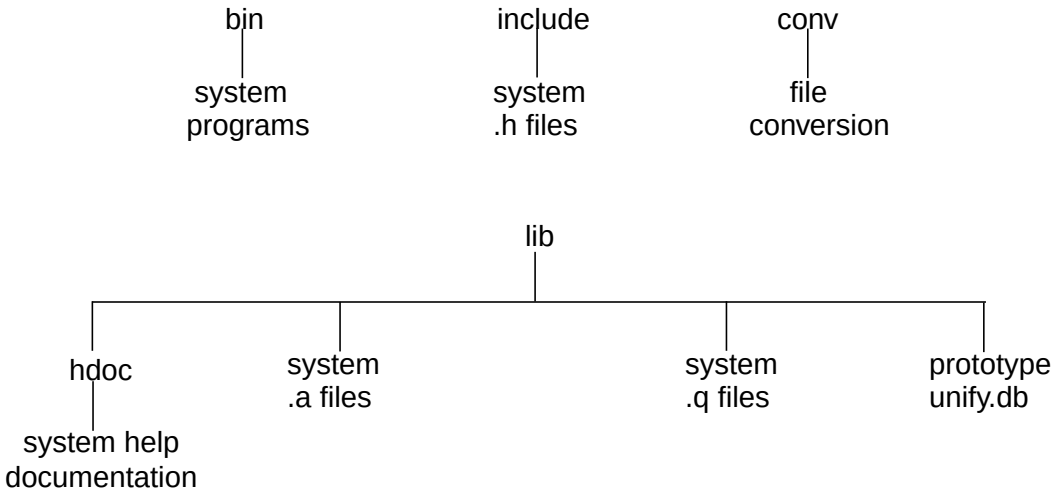


Figure 4.1 Unify DataServer/ELS Directory Structure

The four directories, bin, include, conv, and lib, are located in the directory where you installed Unify DataServer/ELS. Only the bin and lib directories are used at run time. The include directory is used when compiling programs.

An exception to this structure is the Unify DataServer/ELS startup program, named Unify DataServer/ELS, unify is placed in /usr/bin by the installation procedure so you can start Unify DataServer/ELS without specifying a path.

The programs and files in conv are used to convert from one release to another, when necessary. Refer to the Installation Guide for more information.

A test case application directory is shown in the tutorial directory, Figure 4.2. This directory structure follows the Unify DataServer/ELS directory conventions:

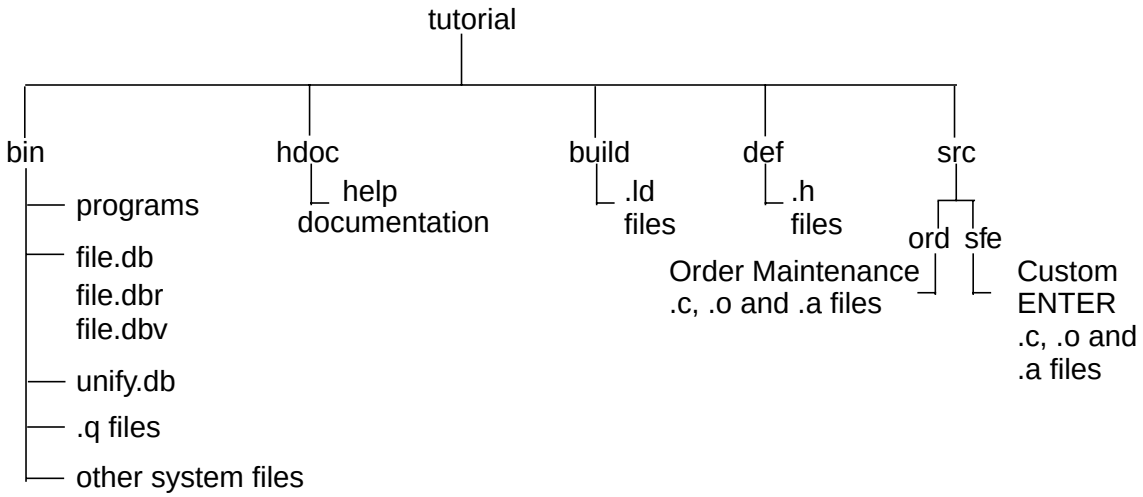


Figure 4.2 Unify DataServer/ELS tutorial Directory

A typical application directory structure looks like the diagram in Figure 4.3. In this figure, xxx and yyy are the names of src subdirectories:

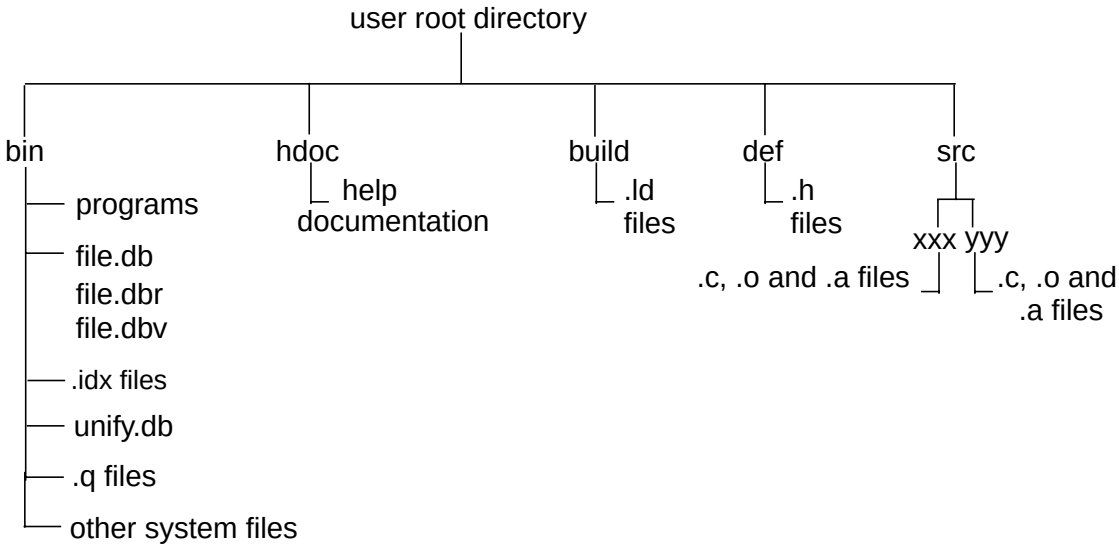


Figure 4.3 Example Application Directory Structure

Chapter 5: Environment variables

Using environment variables

The standard operating system gives you a convenient way of setting up parameters that can be accessed by shell scripts (or batch files) and programs. These parameters, called environment variables, can then adapt shell scripts (or batch files) and programs that reference the parameters to the local environment.

Environment variables take the form:

```
name=value
```

where both name and value are arbitrary strings.

A good example of the use of environment variables is the way the operating system determines which directories to search to find programs. The environment variable named PATH specifies the order in which directories must be searched to find the Unify DataServer/ELS programs.

The default value for PATH is :/bin:/usr/bin or :/usr/ucb. Therefore, the shell searches for programs only in the current directory and in the directories /bin and /usr/bin or /usr/ucb.

Setting environment variables

Suppose on your system, the Unify DataServer/ELS programs are in /usr/unify/bm. You need to add this directory to the PATH list of directories that are searched.

Bourne Shell:

To set the PATH environment variable using the Bourne shell (sh), use the following syntax:

```
PATH=pathname
```

Before the new PATH can take effect in subshells, you must export the name. You do this using the following syntax:

```
export PATH
```

For example, if you are using the Bourne shell, you could set the PATH environment variable as follows:

```
PATH=:/bin:/usr/bin:/usr/unify/bin export PATH
```

C Shell:

To set the PATH environment variable using the C shell {csh), use the following syntax:

```
setenv variable-name value
```

For example, if you are using the C shell, you could set the PATH environment variable as follows:

```
setenv PATH /bin:/usr/bin:/usr/unify/bin
```

You can include environment variable settings and export commands in your.profile or .login file. Then, the environment variables will be set automatically for you when you log in. Refer to .login or .profile in your operating system documentation for more information.

Environment variables enable you to change the standard directories for executable programs and other supporting files. And by setting DBPATH, PATH, and UNIFY correctly, you can execute your Unify DataServer/ELS application in any directory.

The Unify DataServer/ELS installation procedure builds the Unify DataServer/ELS startup program (unify) in /usr/bin, with the basic Unify DataServer/ELS environment variables set and exported (PATH, UNIFY, and BUDEV). Because /usr/bin is part of the default PATH list, you can type unify from any directory and begin using Unify DataServer/ELS through its built-in menu system.

Unify DataServer/ELS environment variables

The following are the names and descriptions of the Unify DataServer/ELS environment variables:

BINEDIT The name of a binary editor. This editor is used in ENTER for editing binary fields. To edit a binary field in ENTER, press CTRL U when the cursor is on the position reserved for a binary field.

BUBLK The size in bytes of a block used by the backup device. If this variable is not set, the block size defaults to 4096 bytes.

BUDEV The complete path name of the system backup device. This is a diskette drive, a cartridge tape drive, or a 9-track tape drive. There is no default value, so this must be set for any program that uses the backup device. It is usually set up by the installation procedure.

BUDRVR The name of a custom program to be used as the backup device filter. The specified program is a custom version of the BUDRVR program. BUDRVR reads from and writes to BUDEV, a file. If the BUDRVR environment variable is not set, the default backup device filter is BUDRVR. For an example of a custom BUDRVR program, see the *Unify DataServer/ELS Direct HLI Programmer's Manual*.

BUTAPESZ The number of blocks that fit on one tape or diskette on the backup device specified in BUDEV. To ensure a successful backup, this variable must be set. If this variable is not set or is set to zero (0), blocks are read or written until the read or write cannot be completed.

The budb utility uses the BUTAPESZ environment variable to indicate the number of blocks to be written to the media. If BUTAPESZ is not set, budb relies on the operating system to correctly handle the end-of-media condition; however, the end-of-media condition is often not handled correctly by the operating system. In these cases, you must specify BUTAPESZ in the environment; otherwise, the backup may not be recoverable.

Always set BUTAPESZ to a value smaller than the number of blocks that can fit on the media. Allow approximately 100 extra 1k blocks of unused media or 25 extra 4k blocks. To determine the approximate number of blocks that can fit on the backup media and to verify correct handling of the end-of-media condition, take the following steps:

1. Back up the database without setting BUTAPESZ. BUDB will display the number of blocks on the media while backing up the

database. Make a note of how many blocks were written on each tape.

2. Set the BUTAPESZ environment variable to approximately 25 less than the number of blocks written to the first tape.

   For example, if budb indicates that 45654 blocks were written to the first tape, you would set BUTAPESZ to 45629.

3. Back up the database again using the smaller BUTAPESZ. This is a safety copy of your database.

4. Unset the BUTAPESZ environment variable; you can unset BUTAPESZ by logging out.

5. Because restoring the database is a risky operation on a production system, you may wish to make additional backups prior to running redb by using tar or cpio for regular files and dd for raw partition database volumes.

6. Using redb, try to restore the backup created in step 1.

   If this fails, your operating system may not correctly handle end of tape. You should then use the media created in step 3 to restore your database, and always remember to set UTAPESZ before starting your backup.

   An alternate technique is to calculate and set BUTAPESZ for all backups. It can be calculated by taking the capacity of the tape in kilobytes and dividing by 4; reduce this number by 5 to 10 percent to allow for interrecord gaps. Then assign this number to BUTAPESZ.

CURR — The display format for amounts. Up to seven characters can be entered to define the currency amount format. The first three characters are required and the last four characters are optional. Each character in the CURR specification is described as follows:

The first character in the string is reserved for the thousands separator. Enter a comma (,), a period (.), or a blank ( ). This character is required.

The second character is the decimal point. Enter a period (.) or a comma (,). This character is required.

The third character indicates the precision, the number of decimal digits that appear after the decimal point. Enter either a two (2) or a zero (0). If you enter a zero, neither the decimal digits nor the decimal point display. This character is required.

The fourth character indicates the position of currency symbol in the amount display. Use a greater than symbol (>) to display the currency symbol on the right. Use a less than symbol (<) to display the currency symbol on the left. This character is required.

The fifth, six, and seventh characters are reserved for the currency symbol. The currency symbol can be any one, two, or three characters except numeric characters. For example, the currency symbol can be $, DM, or xxx. At least one character is required.

The default currency format is comma-separated thousands, with a decimal point and two decimal digits. For example, if CURR is not set, 100 thousand dollars displays as follows:

```
100,000.00
```

And if CURR is set to:

```
" ,2>xxx"
```

then 100 thousand dollars displays as:

```
100000, 00xxx
```

DATETP

The format in which to accept and display dates from 1 January 1900 to 31 December 2077. This environment variable can also be used to change all brackets that display on menus and screen forms to parentheses.

The default date format is MM/DD/YY, where M stands for month, D stands for day, and Y stands for year. The default format also displays brackets on menus and screen forms. The valid values for DATETP are the following:

	AM	American format, MM/DD/YY and brackets.
	EU	European format, DD/MM/YY and parentheses.
	IN	International format, YY/MM/DD and parentheses.
	US	United States format, MM/DD/YY and parentheses.

If you need to format long dates (four-digit year) from 1 October 1752 through 31 December 9999, use the LDATEFMT environment variable, also described in this section.

DBNAME — The name of the database file. This variable need not be set if the default file name file. db is suitable for your system.

To set the database file to something other than the default, use an ordinary file name, not a path name. Therefore, the value of DBNAME cannot contain slashes (/) or backslashes (\).

Unify DataServer/ELS uses the following logic to determine the name of the database file: If unify, db is given to opendb() then unify, db is used,

else if DBNAME is set to a valid file name, then the file named in DBNAME is used, else file.db is used.

If the database file name is not unify, db, the name of the raw database file is obtained by appendng an "r" to the database file name.

DBPATH — The complete path name for the application database files. This includes the files file.db, unify.db, all .idx files, and all .q files.

For example, Unify DataServer/ELS expects to find any user help documentation in the $DBPATH/../hdoc directory. When you set the DBPATH variable, all your application data files must be located in their respective directories, or they cannot be found.

If DBPATH is not set, Unify DataServer/ELS assumes all your application data files are in the current directory. For help

documentation, the usual location is in ../hdoc. Set DBPATH before running Unify DataServer/ELS, or you won't be able to use the application data files.

DEFSIZE The default process size used when NSEGS is 0 to calculate the shared memory attach address.

The default value for DEFSIZE is IM. You can set DEFSIZE to another multiple of IMDEFSIZE must be set to a multiple of IM.

The following example, sets DEFSIZE to IM:

```
DEFSIZE=1048576
```

NOTE: If DEFSIZE is set too high, you may run into a range of addresses used by the stack.

For more information on how NSEGS and DEFSIZE work together, see the description of NSEGS later in chapter 5.

EDIT The name of a text editor or word processor. The default setting is usually the UC Berkeley vi editor. If you don't have the default editor, or if you have something else you like better, you can change it.

The editor is used by Create or Modify Help Documentation (section 12,4), by Edit SQL or RPT Command Files (section 12.5), by SQL (section 16.3) when editing queries, and by ENTER (section 15.2) for editing text fields.

FLTFMT Format for printing floating point numbers. The default for FLTFMT is %g. You can specify any other valid conversion character for C float variables, for example: %17.9f, %.2f, or %.0f.

LDATEFMT The format in which long dates are accepted and displayed. If LDATEFMT is not set, the default format is MM/DD/YY. The valid entries for this variable are the following:

```
DD/MM/YY or DD-MM-YY or DD.MM.YY
DD/MM/YYYY
MM/DD/YY
MM/DD/YYYY
```

```
YY/MM/DD
YYYY/MM/DD
DD/AAA/YY
DD/AAA/YYYY
```

where D (day), M (month), Y (year), and A (alphabetic month) can be entered in upper or lower case.

To specify that days or months less than 10 display as 1-9 rather than 01 - 09, you can enter D and M. For example, if LDATEFMT is set as:

```
LDATEFMT=M/D/YYYY
```

the first day of June displays as 6/1/1999 rather than

```
06/01/1999.
```

To specify a three-letter month, as in "MAY" or "DEC", use the DD/AAA/YY or DD/AAA/YYYY form. The separator can be a slash (/), dash (-), or period (.).

LMPROMO The maximum number of locks allowed on a database object before it is promoted to a like lock of the next higher object, for example, from a record to a table. The Lock Manager uses LMPROMO to conserve memory.

If conflicting locks exist, lock promotion may be postponed until a subsequent lock call.

The default maximum number of locks is 50. You can set LMPROMO to a number from I up to the maximum integer value allowed by your computer system. A larger setting increases multi-user access ability, but uses more memory. A smaller setting decreases multi-user access ability, but uses memory better.

The Lock Manager promotes the locks if there are no conflicting locks. A premature promotion causes other transactions to be locked out.

NOTE: LMPROMO is only used when a Lock Manager shared memory partition is created. If LMPROMO is changed,

you must remove the shared memory partition and restart the application.

LOCKDIR The named pipes directory. The nonshared version of the Lock Manager creates special files called named pipes (or sockets) for inter-process communication. Named pipes have .p suffixes. If LOCKDIR is set, the named pipes are stored in the specified directory. Otherwise, the named pipes are stored in the default directory, /usr/tmp. If /usr/tmp does not exist, the named pipes are stored in the directory specified in DBPATH.

MAXOPNBTS

The maximum number of B-tree files that may be open at one time. If MAXOPNBTS is not set, the default maximum number of open B-trees is 5.

When MAXOPNBTS is set, the size of MAXOPNBTS affects B-tree performance in the following ways:

- If MAXOPNBTS is set to a number greater than 5, B-tree index searches may be faster than at the default. However, fewer file descriptors are available for user programs.
- If MAXOPNBTS is set to a number less than 5, B-tree searches are slower because ACCELL must repeatedly open B-trees. However, more file descriptors are available for user programs.

NOUMSGS Controls the display of progress messages while data dictionary report programs run. The progress messages include: Selecting, Sorting, and Formatting.

If the variable NOUMSGS is set to T (True), the progress messages do not display. The default value for NOUMSGS is F (False).

The following programs display progress messages when NOUMSGS is set to F:

Print Individual Privileges
Print List of Programs
Print Group Privileges
Print Menus

Print Database Design
Print Screens

NSEGS Indicates whether Unify DataServer/ELS can use a system default attach address for shared memory. Possible values are 0 or 1. The default for NSEGS is 0, which is appropriate for most computer systems.

When NSEGS is set to zero (0), Unify DataServer/ELS calculates an attach address based on the DEFSIZE environment variable value, as outlined below:

- Unify DataServer/ELS calculates an attach address that is comfortably outside the process data space. In this case, shared memory segments are attached relative to the process size.
- If Unify DataServer/ELS cannot attach at the calculated attach address, it steps down through memory, in even steps, until it can attach.

NOTE: Because it steps down in even decrements, Unify DataServer/ELS attaches at the original default, if it can find no other attach address.

- Unify DataServer/ELS looks in the DEFSIZE environment variable for a default process size to use in calculating the attach address.

The default value for DEFSIZE is 1MB. You can set DEFSIZE to another multiple of 1MB if you don't want Unify DataServer/ELS to use 1MB. For more information on DEFSIZE, see the DEFSIZE description earlier in chapter 5.

When NSEGS is set to 1, Unify DataServer/ELS uses the default system attach address. Set NSEGS to 1 only if the system provides the same default address for all processes attaching to a segment. Check the hardware documentation for your computer to determine whether a system default address must be used.

PAGELN The page length for LST and RPT reports. The default value is 66. This value is also used for the following data dictionary and database reports:

Print Database Design
Print Menus
Print Group Privileges
Print Individual Privileges
Print List of Programs
Print, Display Database Statistics
Print, Display B-Tree Statistics

PATH — Must contain the directory where the Unify DataServer/ELS system programs are located. In addition, it can contain the directory where your application programs are located, if you want to execute them from somewhere other than the current directory.

PATH — The standard PATH is set to :/bin:/usr/bin.

SEPARATOR — The field separator for RPT, SQL, DBLOAD, and BMTOFF. The default separator is the pipe symbol (|). (BMTOFF is the program that interfaces ENTER to RPT.)

SHMID — The ID number to be used for the shared memory segment dedicated to the Lock Manager. The default setting for SHMID is 4903.

Use SHMID if you want to run an application that uses a different database with a different memory segment.

To prevent shared memory fragmentation, assign a unique ID number to each database; or, specify separate segments for each database.

NOTE: All applications that use the same database must use the same ID number.

SHMMAX — The maximum size allowed for the shared memory segment. Set this variable to a size appropriate for your operating system. At the first call to the Lock Manager, the Lock Manager tries to create a shared memory segment of the requested size. If the requested size is larger than the size allowed by the operating system, the Lock Manager cycles down in 2 K-byte jumps until it successfully creates a shared memory segment. The default setting for SHMMAX is 32767.

SHMMIN — The minimum shared memory segment size. The Lock Manager uses SHMMIN with SHMMAX. The Lock Manager must be able to find a shared memory segment equal to the setting of SHMMIN for Unify DataServer/ELS to run. The default setting for SHMMIN is 4096 bytes.

SPOOLER — The name of your system's line printer spooler. You should set this to the standard operating system spooler, usually lpr.

If you want, you can also specify options for the spooler. For example, if you want to get a mail message after the file is printed, you can set SPOOLER as follows:

```
SPOOLER= "lpr -m"
```

This environment variable must be set so that output can be piped or redirected to $SPOOLER using one of the following syntax forms:

```
cat filename \ $SPOOLER
$SPOOLER < filename
$SPOOLER filename! filename2
```

If, however, you would prefer to send output to an ordinary ASCII file instead of to a printer, you can set SPOOLER as follows:

```
SPOOLER=lprf
```

where lprf is an executable shell script as follows:

```
cat $* > lprf.out
```

SQLPMEM — The amount of memory used during an SQL sort to hold projected fields (the fields in the select clause that are not part of the order by clause). If this variable is set too low and the sort needs more memory for projected fields, the overflow is stored in a disk file. This slows down sort performance. The default value is 0. For more information about setting internal SQL table sizes, see SQL Extensions, section 16.3.

SQLSMEM — The amount of memory used during an SQL sort to hold the sort keys. If this variable is set too low and the sort needs more room for the sort keys, the overflow is stored in a disk file. This slows down sort performance. The default value is 16384 (16k). For more information

about setting internal SQL table sizes, see SQL Extensions, section 16.3.

TERM — The type of your terminal, as defined in your termcap file. There is no default value, so this variable must be set for your terminal to work properly.

Set the TERM environment variable to your terminal type and speed. For example, using the Bourne shell, you set the terminal type and speed with the following syntax:

```
TERM=XXXJC
export TERM
stty nnn
```

where xxxx is the code for your terminal type in the termcap file, and nnn is the baud you are running at (usually 9600). If your terminal requires delays to function properly, you must set the terminal speed explicitly as shown.

The method for setting terminal speed varies depending on the terminal you are using. Read the Users' Manual that came with your terminal for more information.

TERMCAP — The path name of the termcap file. This file tells Unify DataServer/ELS how your terminal works. It may not be necessary for you to set this variable.

The default setting for TERMCAP is /etc/termcap.

TIMEM — The maximum amount of memory used as a buffer for a temporary index. If this variable is set too low and the temporary index requires more memory, the overflow is stored in up to two disk files. If the temporary index uses less memory than specified in TIMEM, only the smaller amount is used. The default value is 8192 (8k).

For more information about setting TIMEM and other internal SQL table sizes, see section 16.3.

UCFLAGS — Specifies the C compiler options to be used by ucc to generate an object file during program loading. If UCFLAGS is not set, the default is

-c. For more information about ucc, see section 1.5 of the *Unify DataServer/ELS Direct HLI Programmer's Manual*.

UCNAME — Specifies the name of the C language compiler to be used during program loading and by ucc. If UCNAME is not set, the default C compiler is cc. For more information about ucc, see section 1.5 of the Direct HLI Programmer's Manual.

UNICAP — The complete path name of the unicap file. If UNICAP is not set, the default setting for UNICAP is the unicap file located in the directory specified by Unify DataServer/ELS.

This file is used by Paint Screen Forms (section 13.2) and the Menu Handler (chapter 3) to determine how the keyboard works. A standard unicap file is supplied with Unify DataServer/ELS. The unicap file is described in section 6.2.

UNIFY — The location of the Unify DataServer/ELS system lib directory. If it is not set to the right directory, or if it is set to null, Unify DataServer/ELS cannot locate the system .q files, the system help files, or the system libraries.

UNIFYTMP — The temporary directory used by Unify DataServer/ELS.

The default value for UNIFYTMP is the directory /tmp. You can reset the temporary directory to any existing directory on your system, provided the following:

- The directory path name begins with root (/)
- The directory path name does not exceed 17 characters
- The temporary directory must also have read, write, and execute permission for all users

The following is an example of a UNIFYTMP setting:

```
UNIFYTMP==/usr/mytmp
```

UUACL — The Unify DataServer/ELS user's access level for the current program. The access level is part of the Unify DataServer/ELS security system and indicates the user's access level for the current program. Its value

is a string of length 2, "00" to "15". This value is defined in Add or Modify Individual Privileges.

UUID — The ID of the current Unify DataServer/ELS user as defined in Add or Modify Individual Privileges. This variable is set by the Menu Handler so user programs can reference it.

VOLPATH — The complete path name for your database volumes. When VOLPATH is not set, all your database volumes are assumed to be in the current directory or in $DBPATH, if DBPATH is set.

NOTE: Only set the VOLPATH environment variable when database volumes are not in the same directory as your application database files.

In addition to these environment variables, Unify DataServer/ELS has two sets of environment variables specific to SQL and RPT. For information on SQL environment variables, see Setting Internal SQL Table Sizes, in section 16.3. For information on RPT environment variables, see RPT Internal Tables and Setting RPT Environment Variables, in section 17.10.

Environment variables and Unify DataServer/ELS programs

You need not worry about setting the Unify DataServer/ELS environment variables if you exit to the shell from a Unify DataServer/ELS menu to run a program, or execute programs by selecting them from a menu. The Unify DataServer/ELS environment variables are available to all unify programs. The definitions of environment variables also have precedence ovei the configuration settings specified in a langcap file.

These are the Unify DataServer/ELS programs you might want to execute like this:

Name	Section	Description
DBLOAD	12.6	Database Load. Lets you load data into a data base from an ASCII file.
LST	18	Listing Processor. Lets you produce sorted file lists with totals and subtotals.

Figure 5.1 Unify DatsServer/ELS Programs

Name	Section	Description
RPT	17	Report Processor. Lets you produce sophisticated reports using a powerful, nonprocedural language.
REPLAY	12.1	Transaction logging replay program. Lets you apply the updates contained in a transaction log file to the database.
SQL	16	Query/Data Manipulation Language. Lets you query and update a data base file using a powerful, non- procedural language.
uid	10	The program to echo the current user's operating system ID to the standard output. RPT input section builder.
rip	17.10	RPT input section builder. Simplifies writing RPT scripts for ENTER screens.
ckunicap	6.2	Program to check the syntax of the unicap file.
enter.ld rpt.ld	*Unify Direct HLI Programmer's Manual*	Shell command files. Let you build custom versions of ENTER and RPT that contain your own host language functions.

Figure 5.1 Unify DatsServer/ELS Programs

Chapter 6: Termcap and Unicap

The termcap and unicap files contain information Unify DataServer/ELS uses to work with your terminal and keyboard.

Hardware functions of most CRT terminals are stored in the termcap file. The default location of this file is in the directory /etc/termcap. Your operating system documentation manual should have an explanation of the format and use of this file under termcap.

The unicap file associates Unify DataServer/ELS commands with key strokes. The default location of unicap is in the Unify DataServer/ELS lib directory. You can change this location with the UNICAP environment variable (Section 5, Environment Variables).

6.1 Termcap

The Unify DataServer/ELS termcap file has descriptions for most CRT terminals. Unify DataServer/ELS uses the termcap file to control cursor movement, screens, and some keyboard input.

If you are on a system that doesn't come with a termcap file, you can use the termcap file included with your Unify DataServer/ELS release and located in the lib directory.

Terminal functions

Unify DataServer/ELS can work with just three standard terminal functions: ce, cl, and cm. However, Unify DataServer/ELS can use some unique functions that are not part of a standard termcap entry. You can add these to your termcap to describe additional video characteristics and some keyboard input responses. The following table. Figure 6.1, is a summary of the Unify DataServer/ELS termcap functions and their uses.

Name	Type	Pad?	Description
co	num		Number of columns on terminal; default is 80
li	num		Number of lines on terminal; default is 23
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
cf	str		Clear unprotected fields
cl	str		Clear all
cm	str	(P)	Cursor motion
ps	str		Start protect mode
pe	str		End protect mode
so	str		Begin stand out mode (reverse video or similar)
se	str		End stand out mode
us	str		Start underline

Figure 6.1 Unify termcap Functions

Name	Type	Pad?	Description
ue	str		End underline
sg	num		Reverse on/off is embedded
ug	num		Underline on/off is embedded
rg	num		Reverse underline on/off is embedded
rv	str		Start reverse underline
re	str		End reverse underline
rl	str		Begin reverse low
ru	str		Begin underline low
rw	str		Begin reverse underline low
nm	str		Begin normal video mode
ws	str		Start write protect (half intensity)
we	str		End write protect (restore full intensity)
gs	str		Start graphics mode
gx	str		Exit graphics mode
ga	str		Lower left round corner
gb	str		Upper left round corner
gc	str		Upper right round corner
gd	str		Lower right round corner
ge	str		Lower left square comer
gf	str		Upper left square corner
gg	str		Upper right square corner
gh	str		Lower right square corner
gi	str		+ intersection
gj	str		vertical bar
gk	str		horizontal bar
gi	str		—1 intersection
gm	str		I— intersection
gn	str		T intersection
go	str		L intersection

Figure 6.1 Unify termcap Functions

Figure 6.2 lists terminal functions that are used only by ENTER:

Name	Type	Pad?	Description
Ec	str		Clear entry; default is CTRL Z
Es	str		Begin search; default is CTRL E
Ex	str		Exit ENTER; default is CTRL X
Ub	str		Move backward/upward on screen; default is CTRL U
Ui	str		Display help text for the current field
Uf	str		Move forward/downward on screen; default is RETURN

Figure 6.2 Keyboard mapping functions (ENTER only)

The three functions shown in Figure 6.3 are required to use Unify DataServer/ELS:

Name	Type	Pad	Description
ce	str	(P)	Clear to end of line
cl	str		Clear all
cm	str	(P)	Cursor motion

Figure 6.3 Required terminal functions

Your terminal must be able to use these three operations. The functions must all be present in the termcap entry for your terminal type. The termcap entries for most terminal types already contain cursor motion (cm), clear all (cl) and clear to end of line (ce).

You may also want to set cd (clear to end of display), if your terminal supports it. Unify DataServer/ELS has a capability that erases the screen line by line if your terminal does not support cd. However, if your terminal does support cd, you can achieve maximum performance by defining a cd entry in your termcap file.

Optional terminal functions

The rest of the terminal functions are optional. If your terminal supports protected fields (the attributes cf, ps, pe, ws, and we), it can display screen prompts in low intensity (protected mode) and data values in high intensity (unprotected mode). Data can be erased from the screen in a single operation by clearing all unprotected fields.

Terminals without these capabilities display prompts and data at the same intensity, and require each data field to be erased separately. When used, these five entries must be in a group, because field security can't operate properly if you use some of these characteristics individually and leave some of them out.

If your terminal supports video attributes, specify so, se, us, ue, rv, re, rl, ru, rw, and nm. You can then display data entry prompts on screen forms in reverse, underline, or reverse underline video modes. Refer to Paint Screen Forms, section 13,2, for information on how to use these features. In prompts, the escape sequences ~r and ~s turn reverse video on and off, ~u and ~v turn underline on and off, and ~w and ~x turn reverse underline video on and off respectively.

If your terminal uses embedded attributes; *i.e.*, turning video attributes on or off takes up a display position, you must specify sg#1 and ug#1.

If you want to use a terminal with 132 columns, specify co#132.

The li attribute should be set to 1124. The Menu Handler and PAINT can use a larger value, but ENTER requires that li either be set to 24 or 0 (zero). You can design a screen with a field below line 22 or 23, but remember that ENTER writes over any lines below 22.

If your terminal supports a graphics character set, specify gs, gx, and ga through go. This lets the Menu Handler draw a box around the current menu.

Keyboard mapping

The entries Es, EC, Ex, Uf, Ui, and Ub let you substitute different control characters for the standard ones used by ENTER screens. You can customize these control characters. The termcap file entries described in the table above must be in the form:

```
name=^X
```

where name stands for the two-character code of one characteristic in the table, and ^X is a control function formed by pressing CTRL and a letter key. For this entry, uppercase letters other than H, J, or M can be used.

The control characters beginning with E are used to set control keys for ENTER. When you use ENTER, a message tells you which keys are understood. The defaults for *search, clear,* and *exit* are CTRL E, CTRL Z, and CTRL X respectively. These defaults are used if the termcap entries are incorrect or not in the termcap file for your terminal.

A Uf control code indicates that another control key will perform the same function *as carriage return does in Unify DataServer/ELS. This gives you a separate* next field key, in addition to the usual carriage return. For example, if you add Uf=^A to your termcap file, CTRL A acts just like carriage return when you are using Unify DataServer/ELS screen forms.

The Ui control code is used to display the help file associated with the current field. Help files are assigned using the Advanced Field Attributes option. The default value is ^I. See Advanced Field Attributes, Section 8.6, for more information on assigning help documentation to fields.

The Ub control code is used to assign a different value for the control key to return to the previous field or prompt. The default is CTRL U, but the Ub parameter lets you change it.

Setting up a termcap file

If you are on a system that does not come with a ternicap file, you should set up a file that contains only entries for terminal types currently used on your system. Start with the Unify DataServer/ELS termcap file, and move the terminal function entries you need to the top of the file. This is because the termcap file is searched linearly for a specified terminal type. The search is considerably faster if you reduce the amount of data to be passed.

6.2 UNICAP

The basic element of the unicap file is a line. Lines have the following form:

command-name = command-key-list {comment]

where valid *command names* are those recognized by Unify DataServer/ELS, and the *command key list* takes one of the forms discussed in the next paragraph. *Comments* are optional.

A command key list is composed of either a single *command key* or several command keys separated by vertical bars (|). Each command key is composed of character representations from one or more of the following character classes: *print characters, escape characters, control characters, octal characters,* and i*maginary characters.*

Some command names allow only a single command key. Command keys cannot contain internal spaces or tabs, except as explained in the following character class descriptions. This allows spaces and tabs to be used in the command key list to improve readability.

Character classes

The following describes the character classes:

print character Any printable ASCII character (upper and lower character case letters, digits, and special characters), excluding the space, caret (^), backslash (\), and "less than"(<). These characters are represented by their normal printed form. Caret, backslash, and "less than" have special meanings in their normal printed forms, so they must be represented as escape characters.

escape character
A representation of a nonprinting special character character. This

includes caret, backslash, and "less than." Escape characters are represented as shown in Figure 6.4.

Representation	Meaning
|E	Escape
\n	New line
\r	Return
\t	Tap
\b	Back Space
\f	Form Feed
\^	Caret
\\	Backslash
\<	Less Than

Figure 6.4 Escape characters

control character

A representation of a possible control key. character Control characters are represented as illustrated in Figure 6.5

Representation	Meaning
any	**letter from A-Z**
^[	CTRL [
^]	CTRL]
^'\	CTRL \
^^	CTRL
^	CTRL^
^_	CTRL_

Figure 6.5 Control characters

octal character

A representation of a nonprinting character that character cannot be represented any other way, or a printing character that has a special meaning when represented as itself. Octal characters are composed of a backslash followed by three octal digits (*octal digits* range from 0 to 7). \000 is not a valid octal character. The space character must be

represented as an octal character, because literal spaces are ignored in the unicap file. Its octal representation is \040.

imaginary character

A representation of nonstandard character sequences character emitted when such non-ASCII terminal keys as "right arrow" are pressed. Different terminal types emit different sequences of octal codes when their keys are pressed.

For keys such as right arrow, left arrow, home cursor and Fl (function key 1), there are no standards. For example, VTIOO terminals emit the octal sequence

'033 133 103' (ESC [C)

for right arrow, while TeleVideo 912 terminals emit a simple

'014' (CTRL L)

If you want to use one of these keys, you must tell Unify DataServer/ ELS how to recognize which nonstandard key has been pressed.

To do this, translate each character sequence emitted by the terminal's physical keys into a standard internal form that Unify DataServer/ELS programs recognize. The Standard internal form for each character sequence is referred to as an *imaginary character*. Each *imaginary character* has a name that refers to it in the unicap file. The maximum length of the imaginary character sequence is six characters.

NOTE: Imaginary characters can be used as command keys in the unicap file only if their names are enclosed in the symbols < and >; for example, exit == <f3>. The name and description of each imaginary character is shown in Figure 6.6.

Name	Description
del_char	This character should be mapped to the sequence produced by the terminal's *delete character* key.

Figure 6.6 Imaginary character command keys

Name	Description
del_line	This character should be mapped to the sequence produced by the terminal's *delete line* key.
down_arrow	This character should be mapped to the sequence produced by the terminal key labeled with an arrow pointing down,
f0, f2,...f20	These characters should be mapped to the sequences produced by the terminal's function keys. There are 21 imaginary function key characters, with names from fO through f20.
help	This character should be mapped to the sequence produced by the terminal key labeled HELP, if your terminal has this key.
home	This character should be mapped to the sequence produced by the terminal key labeled HOME or HOME CURSOR.
ins_char	This character should be mapped to the sequence produced by the terminal's *insert character* key.
ins_line	This character should be mapped to the sequence produced by the terminal's *insert line* key.
left_arrow	This character should be mapped to the sequence produced by the terminal key labeled with an arrow pointing left.
next_pg	This character should be mapped to the sequence produced by the terminal's *next page* key.
Prev_pg	This character should be mapped to the sequence produced by the terminal's *previous page* key.
right_arrow	This character should be mapped to the sequence produced by the command key labeled with an arrow pointing right.
up_arrow	This character should be mapped to the sequence produced by the command key labeled with an arrow pointing up.

Figure 6.6 Imaginary character command keys

Assigning command keys to commands

Some restrictions apply when assigning command keys to commands:

1. A command can be invoked by more than one character, but any character can only be associated with a single command.

2. Digits cannot be used to invoke commands. This is because digits are used to indicate the command repeat count. The exception is 0, which can only be assigned to the FAINT column 0 command.
3. Only nonprinting characters should be assigned to the escape command. If you assign a print character to the escape command, you cannot enter that print character in a screen form, because entering it cancels input mode.
4. Imaginary character names that appear as members of the command key list must be enclosed in the symbols < and >.

You can check the syntax of the unicap file with a command that can be invoked from the shell. The form of the command is:

ckunicap [-p] {file-name}

where *file-name* is the path name of the unicap file you are using. The optional -p prints the file after the messages. It reports errors in the format and representation of characters, but does not check semantics. Semantic errors are checked and reported by PAINT and the Menu Handler before they start.

The ckunicap program also accepts standard input. If no file name is present, ckunicap expects data from standard input.

The standard unicap file

Figure 6.7 shows part of the standard unicap file:

```
SECTION = menus menu handler

ignoremiss = FALSE     report missing command definitions
more = / | <fl0>
select = \E | <f1>
up = ^u | <up_arrow> | ^k
down = \r | ^j ] \n [ <down_arrow>
home = ^x | <home> | <f2>
previous = ^p | <f3>
clear = ^z ) \177 ) <f4>
exit = ^d | <f5>
help = ? | <f6>
info = ^i | <f7>
redraw = ^r | <f8>
toggle = ^t | <f8>
```

Figure 6.7 Standard unicap file excerpt

The unicap file is organized into *sections*, each describing a group of related characteristics. A section begins with the line SECTION = *keyword*. The keyword can be one of the following:

- system
- menus
- paint
- *a terminal type*

Each terminal type defined in termcap should have a corresponding terminal type section in unicap. These unicap sections define the sequence of characters to expect when each key is pressed, and what imaginary character to associate with that sequence. Note that the sequence can consist of a single character, as in the case of the TeleVideo 912.

The following explains the four sections of the unicap file:

SECTION = system

This section describes overall system characteristics. The only valid command name that can be used in this section is ndelaypad.

ndelaypad = n

where n is a positive integer. For fast bauds (in relation to the processor speed), the integer should be small, and for slow bauds it should be high.

It tells how many "no delay" reads to perform to get each character from the terminal. When set correctly, Unify DataServer/ELS recognizes character sequences from function keys, arrow keys, etc., as a single keystroke.

If 71 is too small, Unify DataServer/ELS responds as if the characters have been entered individually. Therefore, Unify DataServer/ELS cannot recognize these imaginary characters correctly.

At 9600 baud, n is usually 0, while at 110 baud, it is usually 4. On some systems, the settings can be 50 and 200.

SECTION = terminal-type

You can have as many terminal type sections as you have terminals to support. This section is composed of lines associating the imaginary characters with the character sequences emitted by the nonstandard keys. If you don't want to use nonstandard keys, then you can omit the section.

The terminal type must match the value of the TERM environment variable (section 5). For example, if TERM=vt100 is the current environment, then you must have a SECTION = vt100 that describes the real-to-imaginary character mapping for VT100 terminals.

The entries in this section have the following form:

imaginary-character-name = command-key

where you choose the imaginary character name from those listed in the previous imaginary character description. Each command key represents the character sequence emitted by the key that you want to equate with the character name. Equate only one sequence with each imaginary character.

SECTION = menus

This section describes the relationship between the Menu Handler commands and the keyboard. The list of internal command names used by the Menu Handler in the unicap file is shown in Figure 6.8.

Name	Action
ignoremiss	report missing command definitions
more	display more command key sets at the bottom of the screen
select	run the highlighted menu line
up	move menu pointer up
down	move menu pointer down
home	go to home menu
previous	go to previous menu
clear	clear selection line

Figure 6.8 Menu Handler commands

Name	Action
exit	exit Unify DataServer/ELS
help	display general Menu Handler help
info	display information about the highlighted menu line
redraw	redraw screen
toggle	toggle command key line on/off

Figure 6.8 Menu Handler commands

Menu Handler commands are also discussed in section 3, The Menu Handler section.

SECTION = paint

This section describes the relationship between PAINT commands and the keyboard. The list of internal command names used by PAINT in the unicap file is shown in Figure 6.9.

Name	Action
Aldfld	add screen field
app	start append input mode
bot	move to last line
co10	move to column 0
cpdisp	toggle cursor position display ON/OFF
delch	delete current character
delfld	delete current screen field
delln	delete current line
down	move cursor down
endln	move to end of line
escape	escape (end current mode)
exit	exit to the Menu Handler
goto	go to screen coordinates x,y
help	print PAINT help
home	move cursor to home position
ins	start insert input mode
left	move cursor left

Figure 6.9 Paint commands

Name	Action
modld	modify current screen field
nrmlon	start normal video
nxtln	move to next line
nxtw	move to next word
oplna	open line above
oplnb	open line below
prvw	move to previous word
quit	quit paint mode
redraw	redraw screen
rep	start replace mode
right	move cursor right
rvon	start reverse video
rvulon	start reverse underline video
screen	display the system name of the current screen
stln	move to beginning of line
trnfld	transfer current screen field
trnln	transfer current line
ulon	start underline video
up	move cursor up

Figure 6.9 Paint commands

Two additional restrictions apply when assigning command keys to PAINT commands:

1. ignoremiss = TRUE Without this line, PAINT prints a message for each command not assigned to a key. With it, PAINT suppresses the messages. Commands not assigned to a key cannot be performed, except for the *escape* and *quit* commands, which are ESC and q by default.

2. mrkchar = *string*

   This keyword lets you change the default *marker character* that marks the original cursor position for various commands such as *add field, transfer field*, *transfer line*, and *reverse video*. The *string* is a single print character. The default marker character is "~".

   See Paint Screen Forms, section 13.2, for more information on how to use PAINT command names.

Part II: Database design and maintenance

Part II contains three sections that describe how to create and maintain a database.

Chapter 7, Introduction to database design, explains the principles of database design, Unify DataServer/ELS database field types, and the access methods used to retrieve data from a Unify DataServer/ELS database.

Chapter 8, Database design utilities, explains the Unify DataServer/ELS tools for creating and modifying a database, reconfiguring a database, using B-tree indexes, and specifying Advanced Field Attributes for database fields.

Chapter 9, Database maintenance utilities, explains the Unify DataServer/ELS tools for reading and writing database backups, rebuilding the hash table, rebuilding explicit relationships between fields, and defining database volumes.

Chapter 7: Introduction to database design

As a database designer, you want to create a design that faithfully models a real-world situation, can be updated accurately and efficiently, and is easy to modify. You must determine what information to store in the database and which groups of information belong together.

This section describes the principles of database design, emphasizing features and access methods specific to Unify DataServer/ELS. The principles of database design are outlined as follows:

1. Develop a preliminary list of the elements, or fields, you want to store in the database. To develop this preliminary list, you should begin by talking to the prospective users of the database to establish their needs.

   Database fields each have a name, a length, and a data type, such as STRING, NUMERIC, AMOUNT, or DATE. Field data types are described in Field Types in the Database Elements subsection and in Appendix A.

2. A relational database contains a collection of tables, each consisting of a group of related fields.

   Organize the preliminary list of fields into tables. Then subject the tables to a series of refinements. You refine tables by determining key fields and the dependencies among fields, and by eliminating unnecessary redundancies. Refining database designs is explained in Refining the Database Design.

3. Create a diagram of the database. This diagram should show each table and its relationships with other tables. A database diagram can often give you insight into the structure and dependencies among the major elements of the database you are building. The Database Diagrams subsection explains how to create a database diagram.

4. Determine the best way to store and access the tables in a Unify DataServer/ELS database. To do this, first analyze the types of queries and updates you

want to perform. Then compare what you want to do with your database to the Unify DataServer/ELS access methods, to select the best approach.

Unify DataServer/ELS provides four access methods: hashing, explicit relationships, B-tree indexes, and buffered sequential access. The decision on which method to use is based on how often an item is modified or queried, update speed versus query speed, and the size and complexity of the database.

The Data Access Methods subsection describes the Unify DataServer/ELS access methods in terms of the functions they best support.

7.1 Database elements

The first step in database design is to develop a preliminary list of the elements you need to include in the database. These will become the fields in your database tables.

To develop a preliminary list of fields, talk to the prospective users of the database application. Determine what information users need to maintain and report.

Make a list of definitions and common terms you can use. For example, in companies where some people refer to parts as "items" and some refer to parts as "products," users must agree on the term the database will use to refer to parts.

Do not apply any constraints at this stage of database development. Your goal in this step is to make a list that is as complete as possible. In later steps you can refine the database and limit its scope.

Field data types

As you develop the list of fields for your database design, think about the kind of information each field represents. Is it a name, a date, a price, a quantity, or what? When you build your database, you must tell Unify DataServer/ELS what each field's data type is.

For more information about field characteristics such as internal length versus display length and changing field lengths or types, see Appendix A.

Field data type reference table

The reference table shown in Figure 7.1 summarizes Unify DataServer/ELS database field types, their default display length, and when to use each type. A more complete

description of each field type follows in this subsection. For information about internal data types, see Appendix A.

Field Data Type	Maximum Display Length	Possible Uses, Notes
NUMERIC	9 digits	Ages, quantities, numbers for calculators
FLOAT	20 digits, including decimal places	Real numbers that are not amounts, integer numbers over nine digits
AMOUNT	14 digits, including decimal places	Currency amounts Note: You can specify a length up to 11. Unify DataServer/ELS adds a decimal point and two decimal digits, increasing the maximum length to 14.
DATE	8	Dates from 1 January 1900 to 31 December 2077
LDATE	11	Dates from 1 October 1752 to 31 December 9999
TIME	5	Time of day
STRING	256	Fixed-length alphanumeric data, such as names, telephone numbers, addresses, serial numbers.
TEXT	256	Variable length text descriptions, such as annotated bibliographies or product specifications. Note: You can specify a display length up to 256 characters. The actual length of the field is the length of the ASCII text stored in the field.
BINARY	n/a	Binary data, such as digitized photos and sounds, graphic images, telemetry data, machine instructions, and so on. Note: You cannot specify a display length. The actual length is the length of the binary data stored in the field.
COMB	n/a	Multi-part and primary key fields. This field's length is the total length of its components.

Figure 7.1 Field type reference table

Field data type descriptions

As you develop your list of fields, also think about how you might want to display data. Although each data type has a default display format, you can set environment variables to control display formats for dates and amounts, and to specify editors for text and binary fields.

For more information about setting environment variables, see chapter 5.

Unify DataServer/ELS recognizes several different data types, each of which serves a specific purpose. The following briefly explains the data types recognized by Unify DataServer/ELS, and gives suggestions on when to use each type.

NUMERIC — This is an integer number with a display length up to 9 digits. A null value is stored as 0. Use a NUMERIC data type for a quantity, age, or other set of discrete units that you want to count or maintain statistics on.

If you need to display codes that have leading zeros or embedded dashes, use a STRING data type instead. For example, if you try to enter 0801 in a NUMERIC field, the 0801 converts to 801. If you try to enter something like 541-321, Unify DataServer/ELS won't let you enter the dash. You can enter only a positive or a negative integer (e.g., 541 or -321).

FLOAT — This is a real number with a display length up to 20 digits, including decimal point. A null value is stored as 0.0.

For example, a FLOAT field display length of 170 specifies 17 integer digits to the left of an implied decimal point. A FLOAT field length of 179 specifies nine digits to the right of the decimal point, a decimal point, and seven digits to the left of the decimal point, for a total display length of 17 characters.

Use a FLOAT data type when you need to store a large integer number or need to store to several decimal places, as in a scientific measurement.

AMOUNT — This is a number used to record a monetary amount. An AMOUNT field's display length can be up to II integer digits (Unify DataServer/ELS adds a decimal point and two decimal digits). A null value is stored as .00.

You can specify how AMOUNT fields display by setting the CURR environment variable. This lets you specify the following:

Thousands separator-comma (,), period (.), or blank

Decimal point (.) or comma (,)

Currency symbol position, (left or right of the amount)

Currency symbol (up to three characters, such as $ or DM).

DATE — This is a short date, with a display length of 8 characters. Its default format is MM/DD/YY, but you can set the DATETP environment variable to specify a different order for month, day, and year. A null value is stored as **/**/**.

Use a DATE data type for a date between 1 January, 1900 and 31 December 2077. This data type takes half the physical storage space of the LDATE data type.

LDATE — This is a long date, with a display length of 11 characters. Its default format is MM/DD/YY (DD/MMM/YYYY in PAINT to permit three-letter months). You can set the LDATEFMT environment variable to specify a different order for month, day, and year. You can also specify a different type of month (numeric or alphabetic) and year (2 or 4 characters). A null value is stored as **/**/**.

Use the LDATE data type for dates between October 1, 1752 and December 31, 9999. That is, use LDATE fields for any long-term events, like business plan data forecaster into the next century, or historical records that span centuries.

TIME — This is a time of day, with a display length of five characters. Its display format is HH:MM. Based on a 24-hour clock, HH is the hour from 0-23, and MM is the minutes past the hour from 0-59.

STRING — This is fixed-length alphanumeric data, up to 256 characters in length.

Use STRING data types for names, addresses, short descriptions, telephone numbers, zip codes, serial numbers, or any combination of characters you don't expect to do calculations on.

TEXT — This is variable-length alphanumeric data that can exceed 256 characters in length, such as a word-processed ASCII file. Although

you can only specify a display length up to 256 characters, a TEXT field's internal length is the length of the ASCII data.

The maximum internal length of a TEXT field is limited only by hardware considerations such as disk size. As data is added or erased, the internal length increases or decreases automatically.

The minimum internal length of a TEXT field is 12 characters in the file.db record. The 12 characters specify the location of file.dbv, which contains the actual TEXT field data.

In general, short alphanumeric fields are stored more efficiently as STRING fields, because of the additional disk I/O and processing overhead associated with TEXT fields.

Unlike STRING fields, which are fixed-length and include trailing blanks, TEXT fields contain only the characters you enter. TEXT fields do not contain trailing blanks unless you explicitly enter the blanks. For more information about editing a TEXT type field, see Entering Data in a Text Type Field, in section 15.2.

Use TEXT data types for such things as annotated bibliographies, where you want to store long descriptions that you can index on and search through for specific references.

BINARY

This is binary data. You can store field information in binary files. The display length of a BINARY field is zero (0), but the internal field length is the length of the binary data.

The internal length of a BINARY field is limited only by hardware considerations such as disk size. A BINARY field's internal length grows or shrinks as data is added or deleted. The minimum length of a BINARY field is 64 bytes.

For more information about editing BINARY fields, see Entering Data in a Binary Type Field, in section 15.2.

Use BINARY data types to store data such as bit maps for graphic screen displays, digitized photos, sounds, instrument readings, machine instructions, and so on.

COMB is a special type of database field that is made by combining one or more other fields. Each of its component fields has a data type specified. A COMB field is merely a reference and does not contain or accept input data. Data is accessed by the name of each component field.

You can use a COMB field type to build a single primary key for a table that needs a key made up of two or more smaller fields.

You can also use a COMB field to divide a field into subfields. For example, the field SSN (Social Security Number) can be set up as shown in Figure 7.2:

Field	Key	Ref	Type	Len	Long Name	Comb Field
SSN			COMB		Soc_Sec_Number	
SSN123			NUMERIC	3	Soc_Sec_123	SSN
SSN4S			NUMERIC	2	Soc_Sec_45	SSN
SSN6789			NUMERIC	4	Soc_Sec_6789	SSN

Figure 7.2 Example COMB field

7.2 Refining the database design

Organize your preliminary list of fields into tables. Each table should contain fields that deal with a single subject, such as inventory items or employees.

For each table, determine which field or fields should be used as the primary key, if a primary key is needed. The primary key is the smallest set of fields that uniquely identifies one record in the table.

For example, suppose you have a preliminary design for a warehouse application. The warehouse stocks a number of items for sale, each made by a particular manufacturer. For each manufacturer, you must know its ID number, city, and shipping status. For each item, you must know its manufacturer, serial number, and selling price.

Your preliminary design contains a table with the following fields:

manfitem	
manf_ID	The ID number of the manufacturer
city	The city where the manufacturer is located
status	The relative ease of getting a shipment from a manufacturer
ser_no	The serial number of the item for sale
price	The price charged for the item

Design 1

In Design 1, each item has only one manufacturer, but each manufacturer can make several items, so you might expect manf_ID to be the better choice for a primary key. However, in Design 1 manf_ID cannot be a primary key, because it doesn't uniquely identify one record in the table. The same manf_ID appears once for every item the manufacturer makes.

Eliminating multiple values in fields

The first step in refining the database design is to make sure no table contains fields that can have more than one value. In the first design for the example table, manfitem,

a manufacturer may make up to six items identified by serial numbers, as shown in Figure 7.3

manfitem

Manf_ID	city	status	ser_no	price
0001	Lynn	10	101,102,103,104,105, 106	2.00 . . .
0002	Reston	20	101, 102	. . .
0003	Reston	20	102	. . .
0004	Lynn	10	102, 104, 105	. . .

Figure 7.3 Multiple-valued fields in manfitem design 1

Because ser_no may contain more than one value, this table does not follow the rules of relational database design.

To follow the rules of relational database design, tables should be set up so each field contains a discrete piece of information. Visualize each table as rows (records) and columns (fields), as shown in Figure 7.4. Each row contains one item per column.

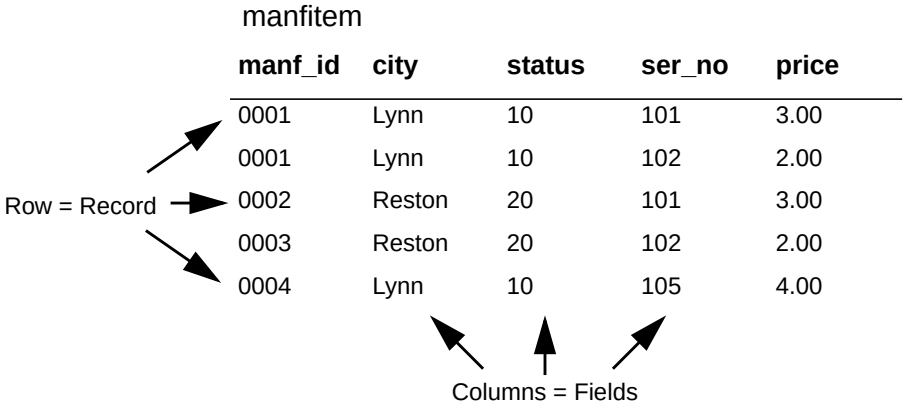


Figure 7.4 Database table visualization

Therefore, to refine the database, the table in Design 1 must be redesigned so the serial number field will never contain more than one value for each record. A possible solution is the table shown in Figure 7.5.

manfitem

Primary Key

manf_ID	city	status	ser_no	price
0001	Lynn	10	101	3.00
0001	Lynn	10	102	2.00
0001	Lynn	10	103	4.00
0001	Lynn	10	104	2.00
0001	Lynn	10	105	1.00
0001	Lynn	10	106	1.00
0002	Reston	20	101	3.00
0002	Reston	20	102	4.00
0003	Reston	20	102	2.00
0004	Lynn	10	102	2.00
0004	Lynn	10	104	3.00
0004	Lynn	10	105	4.00

Figure 7.5 Database Design 2, manfitem table

This refinement, however, means the table now needs both the manufacturer ID and the item serial number to uniquely identify one record. For the record to be unique, the primary key must contain both the manf_ID and ser_no fields. This condition creates a combination primary key.

To determine whether Design 2 is a good relational database design, consider how the design affects the three basic database operations of adding, deleting, and modifying:

1. Adding. Can you add a new record if you don't yet know what items that manufacturer will supply? No, because a combination primary key requires values in each component field. A new manufacturer cannot be added until that manufacturer supplies at least one item. You need both a manufacturer ID and an item serial number to add a record for a new manufacturer.

   For example, you can't add a record for manufacturer 0008 located in Houston, unless you have at least one serial number for an item supplied by manufacturer 0008.

2. Deleting. Can you delete an item that is the only item for a particular manufacturer and leave the manufacturer? No, you cannot delete the item information without also deleting information for the manufacturer's ID, city, and status.

   For example, if you delete item number 102 for manufacturer 0003, you also delete the information that manufacturer 0003 is located in Reston.

3. Modifying. Can you change the location of a manufacturer to a different city? Not easily, because the city for a manufacturer appears in many records. This can cause problems when you are modifying records.

   For example, if manufacturer 0001 moves from Lynn to Revere, you must find every record for manufacturer 0001 and change Lynn to Revere. If you miss even one record, you have manufacturer 0001 located in both Lynn and Revere.

The problem with Design 2 is that too many different kinds of information depend on each other. For example, manufacturer information depends on item information. You should be able to modify manufacturer information without affecting item records, and item information without affecting manufacturer records. These dependencies need to be eliminated.

Eliminating non-key field dependencies

The database in Design 2 obviously needs further refining. Manufacturer information should be separate from item information. To separate the two types of information,

you can divide the manfitem table into two tables, manf and item, as shown in Figure 7.6:

Manf

Primary Key

manf_ID	city	status
0001	Lynn	10
0002	Reston	20
0003	Reston	20
0004	Lynn	10
0005	San Diego	30

item

Primary Key

imanf_ID	ser_no	price
0001	101	3.00
0001	102	2.00
0001	103	4.00
0001	104	2.00
0001	105	1.00
0001	106	1.00
0002	101	3.00
0002	102	4.00
0003	102	2.00
0004	102	2.00
0004	104	3.00
0004	105	4.00

Figure 7.6 Database Design 3, manf and item tables

This revised design overcomes some of the problems involving manf_ID and city in Design 2. The information has not changed, but the design shows some improvements:

1. A new manufacturer can be added even though it does not yet supply any items to the inventory.

   For example, manufacturer 0005 doesn't yet supply an item, but you can add the information that manufacturer 0005 is located in San Diego.

2. The record for an item can be deleted without losing information about the manufacturer.

   For example, the record for manufacturer 0003 and item serial number 102 can be deleted, and the database still has the information that manufacturer 0003 is located in Reston.

3. The location of a manufacturer appears only once in the design. Changing the location of a manufacturer requires only one record update.

   For example, you can change manufacturer 0001's city from Lynn to Revere without having to edit several records.

Yet if you examine Design 3 closely, you should see that it still needs work. Although the item table only contains information about inventory items, the manf table shows a lack of independence among its non-key fields.

For example, the status field depends on the non-key field city, instead of on the key field manf_ID. That is, a manufacturer has a city; and a city has a status. But the status has no real relation to a manufacturer. If the manufacturer's ID number changes, the status is not affected.

However, if a manufacturer's city changes, the status changes as well. This can cause problems.

These dependencies among the manf table's non-key fields cause the following problems for adding, deleting, and modifying data:

1. Adding. You cannot add a record indicating a city's status unless you add a manufacturer for that city. A primary key value is required to add a new record.

   For example, you cannot add the information that Houston has a status of 25 unless you also add a manufacturer located in Houston.

2. Deleting. If the only record for a particular manufacturer is deleted, not only is the manufacturer's information deleted, but also the status information for the city that manufacturer is located in.

   For example, if you delete the record for manufacturer 0005, you lose the information that San Diego has a status of 30.

3. Modifying. The status of a city appears more than once. If you change a city's status, you must change the status for all manufacturers in that city. If you miss a record, you have different statuses for two manufacturers in the same city.

   For example, if you change the status of Lynn from 10 to 20, you must change the status for manufacturers 0001 and 0004.

So the database needs further refinement. To eliminate the problems of status depending on a non-key field, split manf into two tables, manf and cities. The primary key for the new manf table is manf_ID, and the key for cities is city. The new tables have the layout shown in Figure 7.7 .

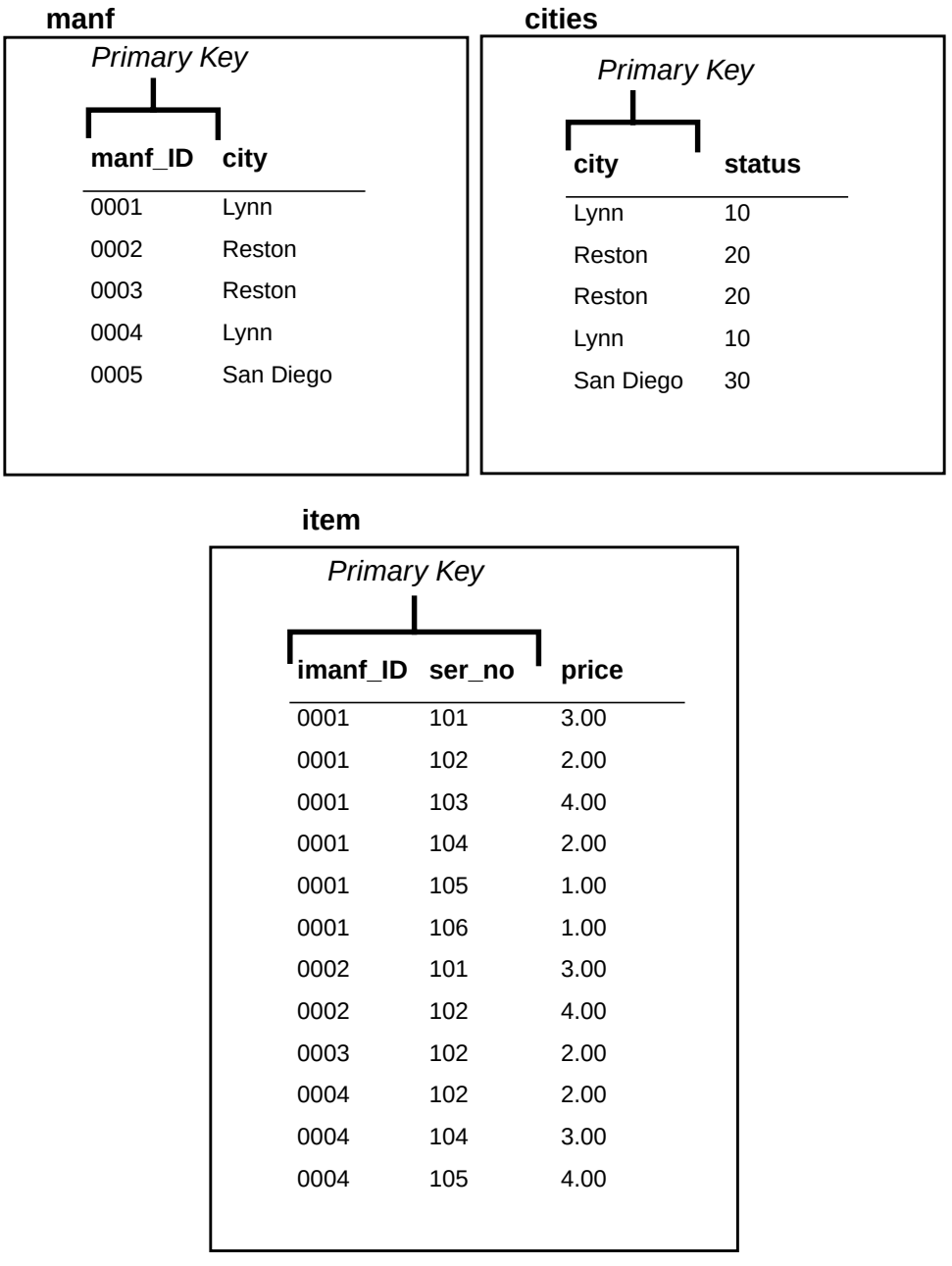


Figure 7.7 Database Design 4, manf, cities, and item table

Now each table in the database design contains information about a single subject. manf contains information about manufacturers; cities, information about city statuses; and item, information about the items for sale.

In Design 4, there is an explicit relationship between mcity in manf and city in cities. The mcity field in manf points to, or references, the key field city in cities. This is shown in Figure 7.8, on the following page, and in Figure 7.9, in section 7.3.

One city can be referenced by several manufacturers. This is a one-to-many relationship. One-to-many relationships are explained in section 7.3, Database Diagrams, and in section 7.4, Data access methods, under Explicit relationships.

Although Design 4 can be expanded, it is a legal Unify DataServer/ELS database design. The design's tables, fields, and relationships are shown in the Unify DataServer/ELS Database design report, Figure 7.8.

DATE: 04/30/99 TIME: 11/29/35 Page 1

DATA DICTIONARY REPORTS
Database Design

TABLE/FIELD	REF	TYPE	LEN	LONG NAME
manf	10			manf
*manf_ID		STRING	4	Manufacturer_ID
mcities	city	STRING	20	City
cities 6				cities
*city		STRING	20	City
status		STRING	2	Status
item 25				item
*item_ID		COMB		Item_ID
imanf_ID	manf_ID	STRING	4	Manufacturer_ID
ser_no		STRING	3	Serial_Number
price		AMOUNT	3	Sales_Price

Figure 7.8 Database design report

For information about database design, see the *Unify DataServer/ELS Direct HLI Programmer's Manual* or any book or article that discusses database design. For a list of suggested sources, see chapter 1, "Introduction to Unify DataServer/ELS".

7.3 Database diagrams

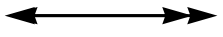
After you have refined your database design, you should draw a diagram showing the tables and the relationships among them. This diagram does not change the definitions of your tables. It merely clarifies the relationships among tables by representing them graphically instead of on several pages of lists.

Relationships can be one-to-one, one-to-many, or many-to-many. For example, in Design 4, one city can be the location of several manufacturers, and one manufacturer can make several items. The tables cities and manf have a one-to-many relationship, and the tables manf and item have a one-to-many relationship.

These relationships can be diagrammed as shown in Figure 7.9:

Figure 7.9 Database diagram, showing explicit relationships

In Figure 7.9, the double-headed arrow indicates "many," and the single-headed arrow indicates "one." Therefore, relationships can be represented using the following symbols:

 indicates a one-to-many relationship

indicates a many-to-many relationship

denotes a one-to-one relationship

An example of a many-to-many relationship is a database for scheduling school courses. Each student can have several teachers, and each teacher can have several students.

The purpose of creating a database diagram is to clarify both your and the users' understanding of the database. A diagram is an easy way to represent your database's major elements and their relationships.

7.4 Data access methods

Refining the database design forces you to ask questions about the functional dependencies in an application. From an initial list of table and fields, and with a recognition of which fields are sensitive to or cause changes in other fields, you refine your design until you have a logically consistent database design.

As you refine your design, you ask questions about how the database will be used, what functions it must support, and what reports and types of queries will be run. The answers to these questions determine which Unify DataServer/ELS access methods must be used.

Generally, Unify DataServer/ELS uses the optimum access method for performance, depending on the type of query or operation being performed. You don't actually have to tell Unify DataServer/ELS to use a specific access method each time it searches for information in your database. But you must design your database so Unify DataServer/ELS has available the most efficient access method for the query or operation.

The Unify DataServer/ELS access methods and the operations they are best suited for are as follows:

Hashing
: This method is used to access records randomly by an exact key. The table must have a primary key. If the primary key is a combination key, all parts of the key must be specified exactly.

Explicit Relationships
: This method is used to join tables that reference one another. A relationship exists when a database table references the primary key of another table.

B-trees
: This method is used when a query must find records based on ranges of values or partial, inexact matches. B-trees are defined using a Unify DataServer/ELS utility.

Buffered Sequential Access

This method is used to access every record in the table, starting with the first record and proceeding one-by-one to the last. This method is usually used with raw files.

The following describes each of the Unify DataServer/ELS data access methods and their advantages and disadvantages.

Hashing (primary key)

Unify DataServer/ELS uses a hash index to search by primary key. A table cannot have more than one primary key, and that primary key must be unique.

In Unify DataServer/ELS, a primary key has two purposes in addition to uniquely identifying a record:

- To let you perform a fast random access (hashing)
- To give you something on which to base an explicit relationship.

However, if you don't need to uniquely identify a record, access by hash table, or specify an explicit relationship, a primary key is optional.

If you need fast random access, use as many fields as necessary to get a unique key. But remember that users must enter every component of the key exactly to find a record. For this reason, primary keys should be short, single fields whenever possible.

If users specify an inexact match or range of values, Unify DataServer/ELS uses another access method to find the record.

For Unify DataServer/ELS to find a record in the hash table, it must search for an exact primary key. Unify DataServer/ELS uses a hash algorithm to calculate the relative record number for the record containing the given key value.

The major advantages of hashing are:

1. It is the fastest random access method when the primary key is entered exactly. It is especially fast for short keys of eight bytes or less.

2. Its speed is relatively independent of the number of records in the database table. A hash table access among a million records takes no longer than among 100 records.
3. The overhead in storage space and database updates is moderate compared to that for explicit relationships and B-trees.

The major disadvantage of hashing is:

1. Hashing does not support access to the data in any special order. For example, hashing is not an effective access method for stepping through manufacturers in alphabetical order by name.

Hashing is appropriate for looking up a unique code like an ID number. Specific examples in the Unify DataServer/ELS Tutorial Manual include the manufacturer number, invoice number, and item number.

Explicit relationships

An *explicit relationship* refers to an access method that supports a one-to-many linkage between two tables. If you need to join two tables frequently, an explicit relationship joins the tables faster than any other access method. Consider the linkage between the manf and item tables from Design 4 shown in Figure 7.10:

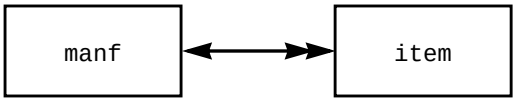


Figure 7.10 Explicit relationship between item and manf

Each manf record has a set of related item records. A linkage exists between two manf and item records if the manufacturer makes the item. In this relationship, the

manf table is the parent, and the item table is the child. Figure 7.11 illustrates this relationship::

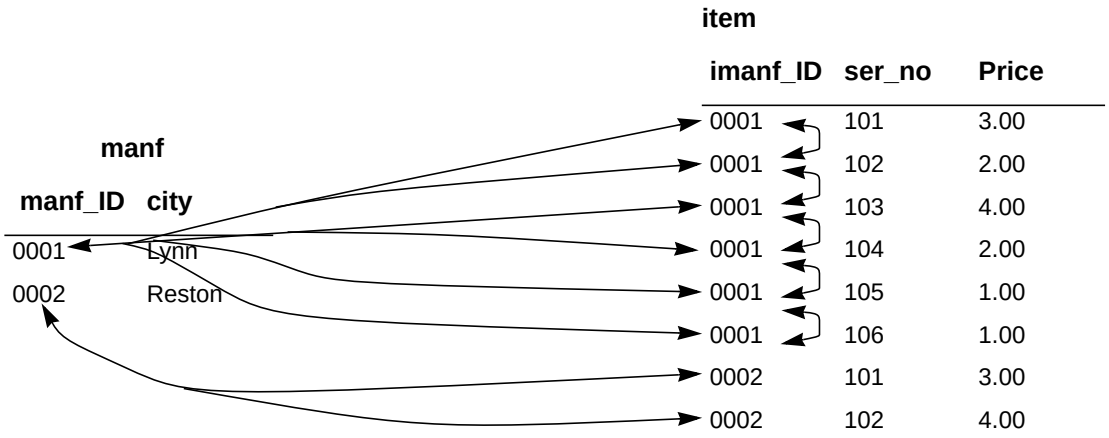


Figure 7.11 Parent-child relationships between manf and item

Besides functioning as a data access method, an explicit relationship also automatically enforces referential integrity. If an explicit relationship exists between two tables, Unify DataServer/ELS only lets you store existing parent key values in the child table. Unify DataServer/ELS does not let you delete a parent record whose key is referenced by child records.

Explicit relationships save storage space and processing time when compared to a B-tree. This is important because a relational database often consists of many small tables. Unify DataServer/ELS must join these tables frequently to process queries. Explicit relationships are the fastest method for this.

In the database design, to declare an explicit relationship between two tables, reference the parent primary key in the REF column of the dependent child field. For example, to declare an explicit relationship between manf and item, place manf_ID, the key for manf, in the REF column for imanf_ID in the item table. Figure 7.12

illustrates this with an excerpt from the Database Design report for Design 4.

TABLE/FIELD	REF	TYPE	LEN	LONG NAME
item 25				item
*item_ID		COMB		item_ID
imanf_ID	manf_ID	STRING	4	Manufacturer_ID
ser_no		STRING	3	Serial_Number
price		AMOUNT	3	Sales_Price

Figure 7.12 Excerpt from report for database design 4

The major advantages of explicit relationships are:

1. Explicit relationships take less disk space and less time to process than do B-trees.
2. Referential integrity is automatically enforced.
3. Given a parent key value, finding and processing each child record is fast and efficient.
4. Given a child record, accessing information about its parent record is fast and efficient.

The major disadvantages of explicit relationships are:

1. Relationship pointer chains are maintained in a LIFO (last-in, first-out) order only, so records cannot be retrieved in a specified sequence without a sort.
2. If you want to add or drop an explicit relationship, you must change the database design and then reconfigure the database.

Properly used, explicit relationships can dramatically improve query performance, at a relatively low cost in terms of disk space and update performance. When joining five or more tables in multi-user applications, explicit relationships can outperform B-trees by a factor of more than ten to one.

B-trees

B-trees have become the standard for indexes in most database management systems for several reasons:

1. B-trees are always balanced, so every search takes the same amount of time.
2. The number of disk accesses required to find a record rises very slowly as the index gets larger.
3. B-trees reorganize themselves automatically, so their performance stays constant even after many additions and deletions.

Data access by B-trees is a modified binary search technique. To understand how B-trees work, it is helpful to review how a binary search operates.

Before a binary search can be used to find records, the records must be sorted. For example, a binary search can be used to find the name "DuPres, Jacques" in a telephone directory. The binary search can find the name only if the list of names has been alphabetized and the name exists. The telephone directory is already sorted, as shown in Figure 7.13.

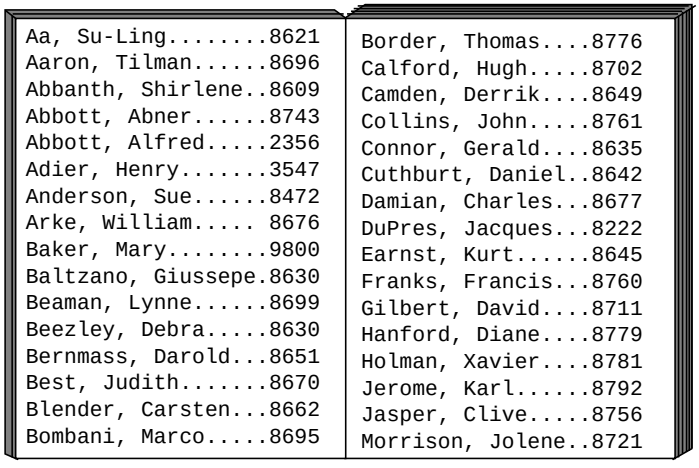


Figure 7.13 Simple telephone directory

A binary search starts by dividing the list of names in the directory into two halves, each containing the same number of names. Suppose that the second half begins with the name "North, David," as shown in Figure 7.14. The first decision in the binary search process is whether "DuPres" occurs *after* "North." The answer is no, it must occur *before*. Therefore, the second half of the directory can be eliminated and the

search restricted to the first half.

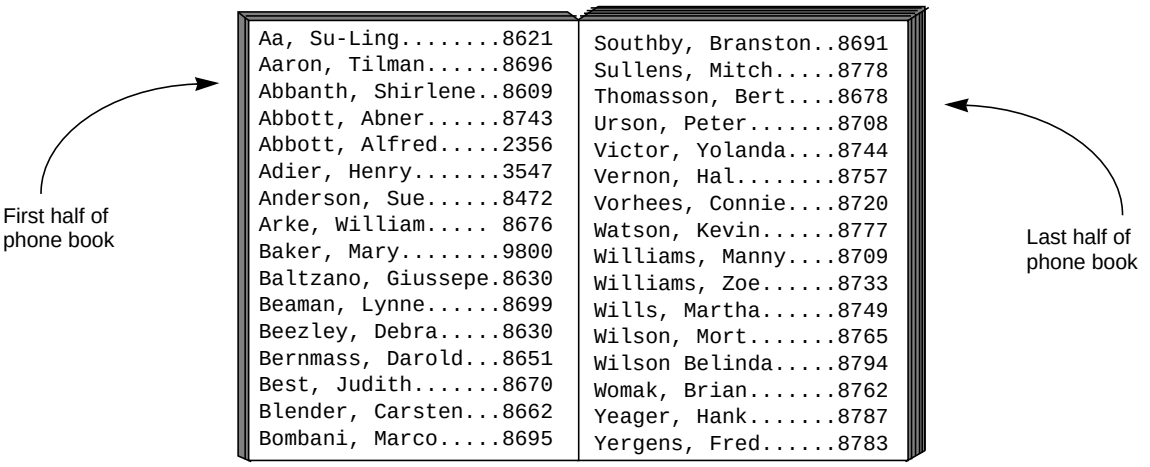


Figure 7.14 Begin binary search of phone book

The divide and search process is repeated for the remaining directory names, as shown in Figure 7.15. The name "Gull, Harold" begins the second half of this new division; *i.e.*, the second quarter of the directory. Because "DuPres" is before "Gull," the second quarter of the directory can be eliminated. At this point, the search space is one-fourth the original size, and only two comparisons have been made.

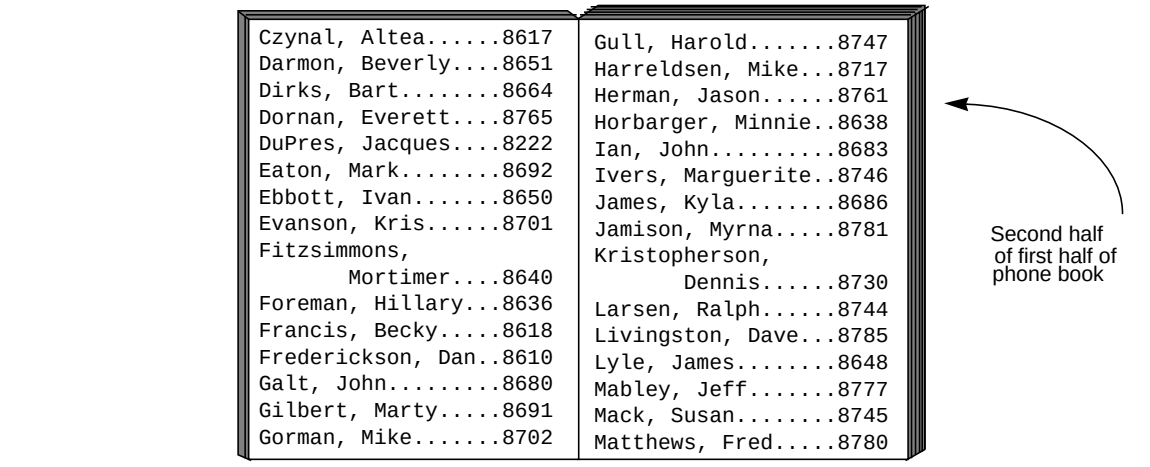


Figure 7.15 Repeat binary search

As the procedure is repeated, the search space is reduced to one-eighth, one-sixteenth, one-thirty-second, and so on, until only "DuPres" is left, shown in Figure 7.16.

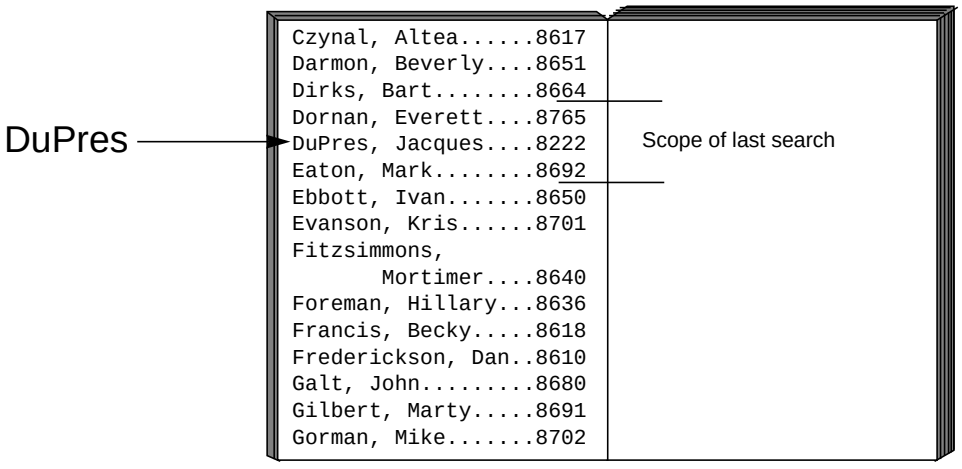


Figure 7.16 Last binary search locating DuPres

A binary search can find one value out of 1024 using no more than 10 decisions. Compared to a sequential search, which requires an average of 512 decisions to locate one value in 1024. A binary search is very efficient.

Even so, some simple modifications can improve the binary search technique. For example, suppose that each decision is a one-of-three comparison. That is, it identifies which third of the search space holds the record you want to find. Figure 7.17 locates the two names that sit on the first and second one-third boundaries in the telephone directory example.

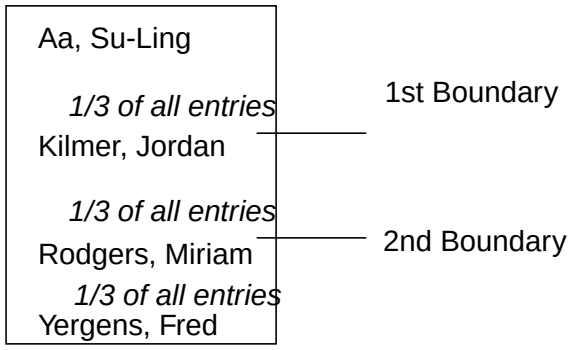


Figure 7.17 A one-of-three decision

The one-of-three decision depends on finding which of the following statements is true:

1. The record is after "Abbott, Alfred" and before "Kilmer, Jordan."
2. The record is after "Kilmer, Jordan" and before "Rodgers, Miriam."

With this one-of-three modification, the records can be located much faster. For example, a search of 1024 records takes no more than seven decisions, instead of the ten decisions required for a binary search.

If a one-of-three search is faster than a binary search, a one-of-four search is faster yet. Similarly, the search gets faster as the N of "one-of-N" gets larger.

The Unify DataServer/ELS B-tree access method uses a one-of-N search, where N is chosen on the basis of the data to be searched, the length of the fields to be indexed, and the disk block size. The search is most efficient when each of the N categories contains an equal number of search objects. B-trees are designed to keep an equal number of entries in each category. They are balanced.

Unify DataServer/ELS lets you build up to 255 B-trees, containing up to eight fields each, within a database. You can create B-trees using the option "Add, Drop B-Tree Indexes."

B-trees excel in three types of queries:

1. Retrieving records in sequence, sorted by the values in the indexed fields.
2. Locating records when the exact value is not known, but the beginning of the value is known. For example, a B-tree can quickly locate all people whose last names begin with "Joh" (as in John, Johnson, and Johnston). However, a B-tree cannot locate people whose last names end with "son" (as in Ellison, Johnson, and Thompson).
3. Locating records in a given range, such as all amounts between $5.00 and $10.00, or all dates between January I and January 31.

B-trees can also be used to speed up joins when you don't want to use an explicit relationship. For example, suppose you have customer and order tables, and the customer number is stored in the order. You can build a non-unique B-tree on the customer number in the order table, and find all orders for a specific customer by searching for the customer number.

The major advantages of B-trees are:

1. B-trees permit ordered access to all records in a given table, based on the values of the indexed fields. When you must access large numbers of records in sorted order very quickly, use B-trees.
2. B-trees speed up queries involving inexact matches, ranges of values, or "begins with" searches. B-trees can be used on any field to create a secondary key, complementing or replacing the primary key.
3. B-trees can be added or dropped without reconfiguring the database.

The major disadvantages of B-trees are:

1. B-trees use more disk space than other Unify DataServer/ELS access methods.
2. Access by B-trees is slower than by hashing or explicit relationships, because B-trees use fairly complex algorithms for searching and balancing.

In most cases, the advantages of B-trees outweigh their disadvantages. However, because of their disk space requirements and processing time, you should establish a clear need before using them. Used correctly, explicit relationships can be used to perform joins much more efficiently.

Buffered sequential access

The most basic access method is sequential access, which scans every record in a table one by one. Unify DataServer/ELS uses two types of sequential access, buffered and nonbuffered.

For nonbuffered sequential access, the operating system reads data from a file one disk block at a time. For buffered sequential access, the operating system reads many blocks of data from a file into an internal buffer, until the buffer is filled.

A file is actually made up of blocks scattered over the disk. The operating system keeps an index of each block's location. When reading from or writing to a particular location in a file, the operating system uses the index to determine which physical block of the disk to use.

Using the index to locate blocks is efficient for files up to one or two megabytes that are accessed randomly and created and deleted often. Such files include text files, program source code, documents, shell scripts or batch files, and executable files.

However, using an index to locate blocks is not efficient for many typical database files. These database files are often larger than ten megabytes, and they often must be accessed sequentially for reports and backups. When a file gets this large, the indexing structure binders performance. To read the correct data block, up to three index blocks must be read first. This can slow down the read process considerably.

The solution for large files is to use the Express I/O file system, often called a raw file. A *raw file* is located in physically adjacent blocks, so the indexing structure is not needed. To create a raw file, unmount one of the file systems, use "Define Database Volumes" (section 9.5) to tell Unify DataServer/ELS about that disk partition, and then perform a "Create Database" (section 8.3).

If the database is made up of raw files, the operating system can transfer raw file blocks directly into a user program's data space. The program can specify a buffer larger than the standard operating system buffer. (The default buffer size is 2048 bytes.) Because raw file disk blocks are physically adjacent, a single disk read can return several blocks of data.

Buffered sequential access can be used for files that are not raw. Disk blocks tend to be scattered in non-raw files, so you must wait between each read for the disk head to find the correct track and then for the correct block to rotate under the head. Using a raw file eliminates these delays.

The major advantages of buffered sequential access are:

1. Buffered sequential access had no disk storage overhead.
2. If used with raw files, a buffered scan is much faster than an unbuffered scan for read-only transaction.

   For more information on raw data file, see the File subsection in chapter 4 and the define Database Volumes subsection in chapter 9.
3. If you must read every record or most records in a table and order is not important, a buffered scan is the fastest method.

The major disadvantages of buffered sequential access are:

1. It looks at every record in the table. If you are only interested in a few of the records, there is no need to read all of them sequentially.
2. The records are returned in a system-defined order. If you need the records in a particular order, you must sort them.
3. If you update most of the records in a table, you negate the advantages of buffering. Each record must be searched for, stored in a buffer, and written to the disk. Nonbuffered sequential access or hashing is more suitable, depending on your needs.
4. Buffered sequential access is designed for reading forward through the file. Although you can read backward using the previous option, this is much less efficient because of how the buffering scheme works.

Chapter 8: Database design utilities

The Database Design Utilities chapter describes the tools you can use to design a Unify DataServer/ELS database. These database design tools are programs that display as options on Unify DataServer/ELS menus. The first program described in this chapter is listed on the Unify DataServer/ELS Main Menu, while the other programs described in this chapter are found on the Database Design Utilities menu.

These Unify DataServer/ELS database design tools are desribed in the following sections:

Section 8.1 Design and create a new database

Section 8.2 Modify database design

Section 8.3 Create database

Section 8.4 Reconfigure database

Section 8.5 Add, drop B-tree Indexes

Section 8.6 Advanced field attributes

Design and create a new database (dbcreate) enables you to specify tables and fields for an initial database. You can also print the database design, create default database screen forms for each table, create the new database, and create a menu containing the new screen forms.

Modify database design (schent) is similar to dbcreate. You can use it to change an existing database's design. This program, however, does not have the options available with Design and Create a New database.

Create database (crdb) creates an empty database without doing all the other things that dbcreate does for you.

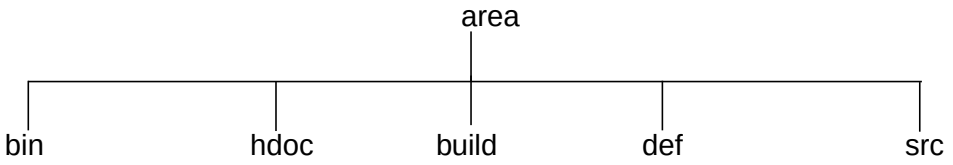
Reconfigure database (scom) reformats the database file and updates the data dictionary after you make changes with Modify database Design.

Add, drop B-tree indexes (idxmnt) adds a secondary index to the database or drops an existing index. This program can also rebuild the existing B-tree indexes.

Advanced field attributes (afa) lets you specify field attributes in addition to the basic attributes of length and data type. You can specify a default value, a list of legal values, an information message, and a help file with more information about the field.

Database directory structure

Your database should be created in a directory structure suitable for the application development that will begin once the database is established. The following diagram suggests a directory structure:



This is not a required structure, but rather an example.

The bin directory contains the data dictionary file, the program executable files, the database file and other supporting files required at run time.

The hdoc directory contains custom help documentation for the programs in bin.

The build directory contains the command files necessary to create the custom executable files.

The def directory contains the files that are "included" in the custom program source code. These files contain system-wide data definitions and parameters.

The src directory contains the source code for the programs. For large systems that contain multiple modules, the src directory should have module subdirectories.

8.1 Design and create a new database (dbcreate)

This Unify DataServer/ELS program is extremely useful in the development of a new database application. The functions performed by design and create a new database are similar to the following Unify DataServer/ELS programs:

Modify database design
Create database
Print database design
Create default screen form
Add, modify or delete menus

This chapter explains the two phases of design and create a new database: the *database design phase* and the *database create phase*.

The database design phase lets you design the database structure. To do this, you first define the tables using the *table entry screen*. After a table has been entered, the program displays the *field entry screen*. You then define the fields for the current table. When done with field entries, you return to the table entry screen to enter the next table.

The database create phase lets you print the new database design, create an empty database from the new database design, create default data entry screens for each table, and create a Data Entry Screens menu.

Database tables

The *table entry screen* in the Design and Create a New database program enforces the rules of database design as you make entries. It reports errors immediately and skips any invalid attributes. If you want to enter an attribute and cannot, you might have to change another attribute of the field to complete the entry.

The table entry screen displays as follows:

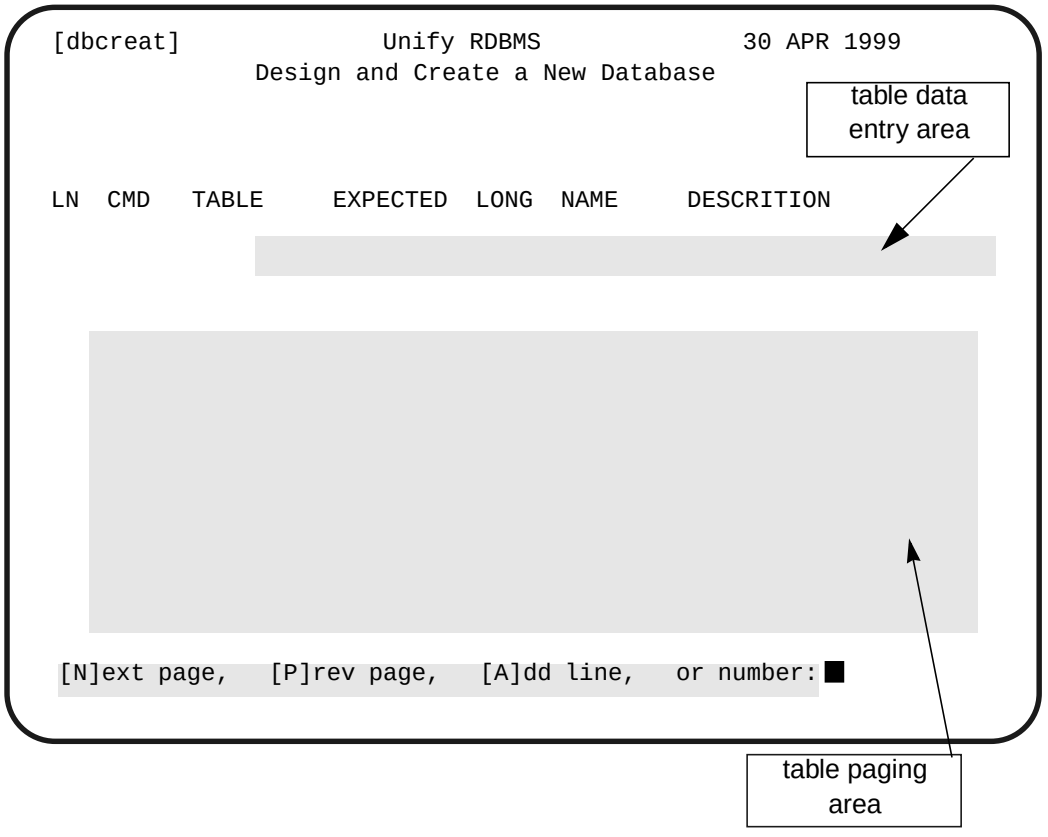


Table information is added in the *table data entry area* of this screen. In the *table paging area* below the data entry area, existing entries can be modified, deleted, or moved. All tables display in a paged format, with II descriptions per page.

The screen prompts and column headings are described as follows:

[N]ext page, [P]rev page, [A]dd line, or number:

The operation prompt. This prompt lets you choose the operation mode for the table entry screen. The purpose of each selection is as follows:

n	Displays the next page of database tables
p	Displays the previous page of database tables

	a	Moves the cursor to the table entry area to let you add new database tables
	1-256	Displays the page that contains the indicated database table, and positions the cursor at that table to let you modify or delete it as explained in the CMD column description.
	CTRL U	Ends the database design phase and starts the database create phase.
LN	A line number assigned by the system. Use it to refer to the table you want to modify.	
CMD	The area used to enter a command to perform an operation on the current line. When the cursor is in this column, it moves up and down as you press CTRL U and RETURN. The position of the cursor marks the current line. Valid commands are as follows:	
	f	Modify the fields for the current table by going to the second page of the screen form
	m	Modify **EXPECTED, LONG NAME**, or **DESCRIPTION** for the current table
	d	Delete the current table
	1-256	Renumber the current table as shown and reorder the tables on the screen form
	q	Redisplay the operation prompt at the bottom of the screen
TABLE	A unique table name of up to 8 characters. The table name can contain only letters (upper and lower case), numbers, and the underscore character (_). It must begin with a letter. This name is used to reference the table throughout the system. While entering data, a CTRL U in this column terminates table entry mode and redisplays the operation prompt.	

EXPECTED The expected number of records in this table. Unify DataServer/ELS lets you exceed this number by 66%; then you must increase it before adding more records. In some cases, it is easier to increase the expected number than to decrease it. If you do want to decrease this number and are having problems, see the steps that accompany the resulting error message (at the end of this chapter).

LONG NAME

A more descriptive table name of up to 16 characters that is used by ENTER to display error messages. It can contain only letters (upper and lower case), numbers, and the underscore character (_). It must begin with a letter.

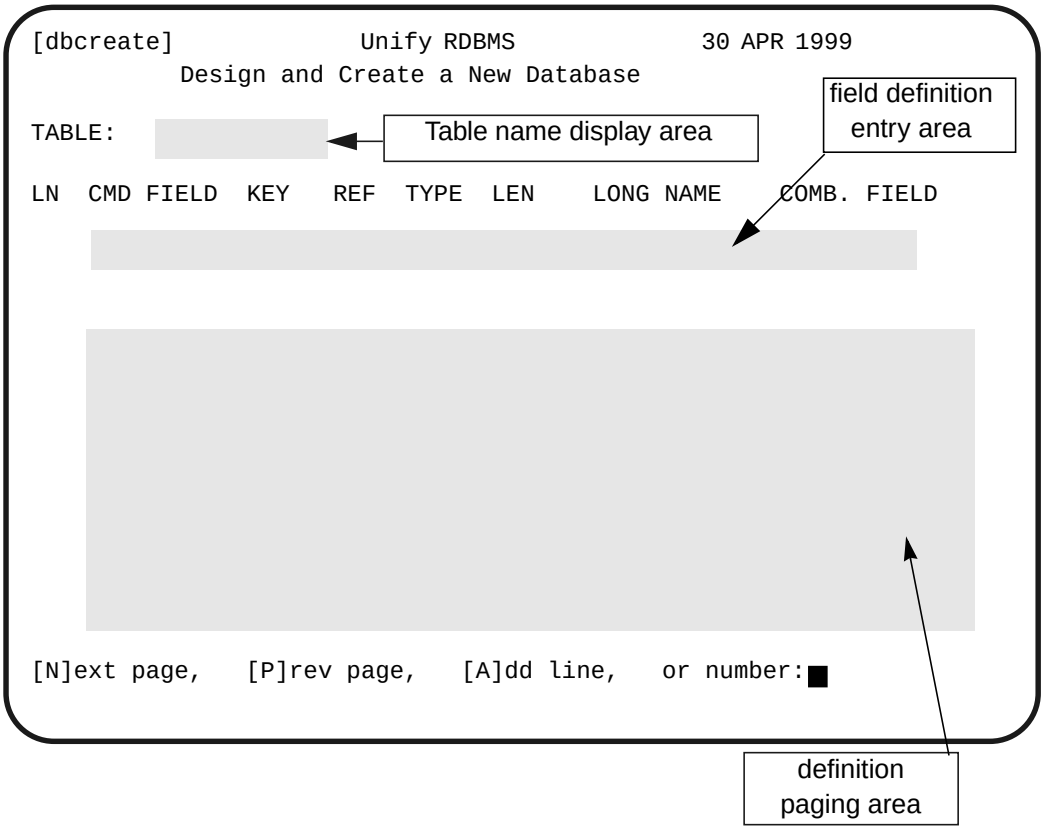
DESCRIPTION

A comment field where you can enter a description of the table. While in the data entry area, a RETURN gets you to the field entry screen.

Database fields

The *field entry screen*, like the table entry screen, enforces the rules of database design as you make entries. This screen form is used to enter or modify a database field description.

The field entry screen displays as follows: .



The screen prompts and column headings are described as follows.

[N]ext page, [P]rev page, [A]dd line, or number

The operation prompt. This prompt lets you choose the operation mode for the field entry area. The purpose of each selection is as follows:

n	Displays the next page of fields for the current table
p	Displays the previous page of fields for the current table
a	Moves the cursor to the field data entry area to let you add new fields

	1-256	Displays the page that contains the indicated field, and positions the cursor at that field to let you modify or delete it as explained in the CMD column description
	CTRL U	Ends the field entry screen and redisplays the table entry screen

TABLE — A display-only prompt that shows you the name of the table you are adding or modifying fields for.

LN — A line number assigned by the system. It is used to reference the line you want to modify.

CMD — The area used to enter a command to perform an operation on the current line. When the cursor is in this column, it moves up and down as you press CTR U and RETURN. The position of the cursor marks the current line. Valid commands are as follows:

m	Modify KEY, REF, TYPE, LEN, LONG NAME, or COMB. FIELD for the current field.
NOTE:	If you modify a field's type or internal length, any B-tree index associated with this field is dropped when the database is created or reconfigured.
d	Delete the current field.
NOTE:	If you delete a field, any B-tree index associated with this field is dropped when the database is created or reconfigured.
1-256	Renumber the current field as shown and then reorder the fields on the screen form
NOTE:	You cannot renumber the primary key field. Also, you cannot renumber component subfields of a COMB field except by renumbering the COMB field.

q — Redisplay the operation prompt at the bottom of the screen

FIELD — A unique field name of up to 8 characters. The field name can contain only letters (upper and lower case), numbers, and the underscore character (_). It must begin with a letter. This name is used to reference the field in several Unify DataServer/ELS options.

While entering data, a CTRL U in the field definition entry area under FIELD ends data entry mode.

KEY — The primary key marker. An asterisk in this column marks the current field as the primary key of the table.

NOTE: TEXT and BINARY type fields cannot be used as primary keys.

REF — The name of the primary key of another table. It establishes an explicit relationship to another table.

TYPE — The data type of the field. For all data types except TEXT type, specify a data type by entering the first character of the type in upper or lower case. Specify a TEXT data type by entering x or te in upper or lower case.

The table in Figure 8.1 summarizes the valid data types, comparing their display and internal lengths.

Data types

Symbol	Name	Display length	Internal length and type
n	NUMERIC	1-4	
d	DATE	n/a	short
t	TIME	n/a	
n	NUMERIC	5-9	
a	AMOUNT	1-7	long
l	LDATE	n/a	
f	FLOAT	any	
a	AMOUNT	8-11	double

Figure 8.1 Table of valid field data types

Data types

Symbol	Name	Display length	Internal length and type
s	STRING	1-256	char[n] (string)
te,x	TEXT	1-256	limited only by system
b	BINARY	n/a	limited only by system
c	COMB	n/a	n/a

Figure 8.1 Table of valid field data types

LEN

The default display length of the field on screen forms and reports. For NUMERIC and STRING fields, LEN is exactly the display length. For FLOAT fields, LEN has the form nnd, where nn indicates the total number of display positions including the decimal point and d the number of digits to the right of the decimal point.

For example, a FLOAT field with a LEN of 123 has a default format of:

nnnnnnnn.nnn

A FLOAT field with a LEN of 100 has a default format of:

nnnnnnnnnn

Note that there is no decimal point, so the format here has 10 digits. For AMOUNT fields, LEN indicates the number of digits to the left of the decimal point, and assumes 2 digits to the right of the decimal point.

The total display length of AMOUNT fields is LEN+3+space required for commas. For example, an AMOUNT field with a LEN of 11 has the following default display format on screen forms and reports:

nn,nnn,nnn, nnn.nn

If the CURR environment variable is set, the amount field displays in the specified format. Space is reserved for commas (or other thousands separator), even if no commas are specified in CURR.

Maximum LEN's for the various field types are as follows:

NUMERIC	9
FLOAT	179
AMOUNT	11
DATE	n/a, defaults to 8
LDATE	n/a,defaults to 11
TIME	n/a, defaults to 5
STRING	256
TEXT	256
BINARY	n/a
COMB	n/a, computed by system

LONG NAME

A more descriptive field name of up to 16 characters. It can contain only letters (upper and lower case), numbers, and the underscore character (_). It must begin with a letter. The long name need not be unique throughout the database as the field name must be. However, it must be unique to this table. The long name may be identical to the field name. If you do not specify a long name, you will not be able to use SQL on your database.

The long names of most database fields are used to generate their access names. For example, two database tables have the fields shown in Figure 8.2.

model				model
*mokey		COMB		Model_Key
monum		NUMERIC	7	Model_Number
monano	mano	NUMBERIC	4	Manufacturer_ID
mdes		STRING	30	Description
item				inventory_item
*sno		NUMERIC	9	Serial_Number
imodel	mokey	COMB		Model_ID
iad		DATE		Acquisition_Date
isal		AMOUNT	5	Sales_Price
iorder	onum	NUMERIC	9	Order_Number

ipamt	AMOUNT	5	Purchase_Price

Figure 8.2 Database tables and fields

Using the two tables. Figure 8.3 shows how the long names of most database fields are used to generate their access names.

For the Field:	Its Access Name Is:
momano	Manufacturer_ID
mdes	Description
isal	Sales_Price
iorder	Order_Number

Figure 8.3 Field access names

However, because mokey is a COMB field with two component fields, momano and monum, two implied fields are created in the imodel field (one for each component field in mokey).

When a field such as imodel references a COMB field in a second table, the field's long name is not used to generate its access name.

The referencing field must have separate access names for each of its implied fields. The access name of each implied field is a combination of the first field's short database name and the short name of each component field it references in the second table. Figure 8.4 illustrates this naming convention.

For the field:	Its implied fields' access names are:
imodel	imodel_monum imodel_momano

Figure 8.4 COMB field access names

COMB.FIELD

This is the name of the combination field (a COMB type field) in the current table of which the current field is a part.

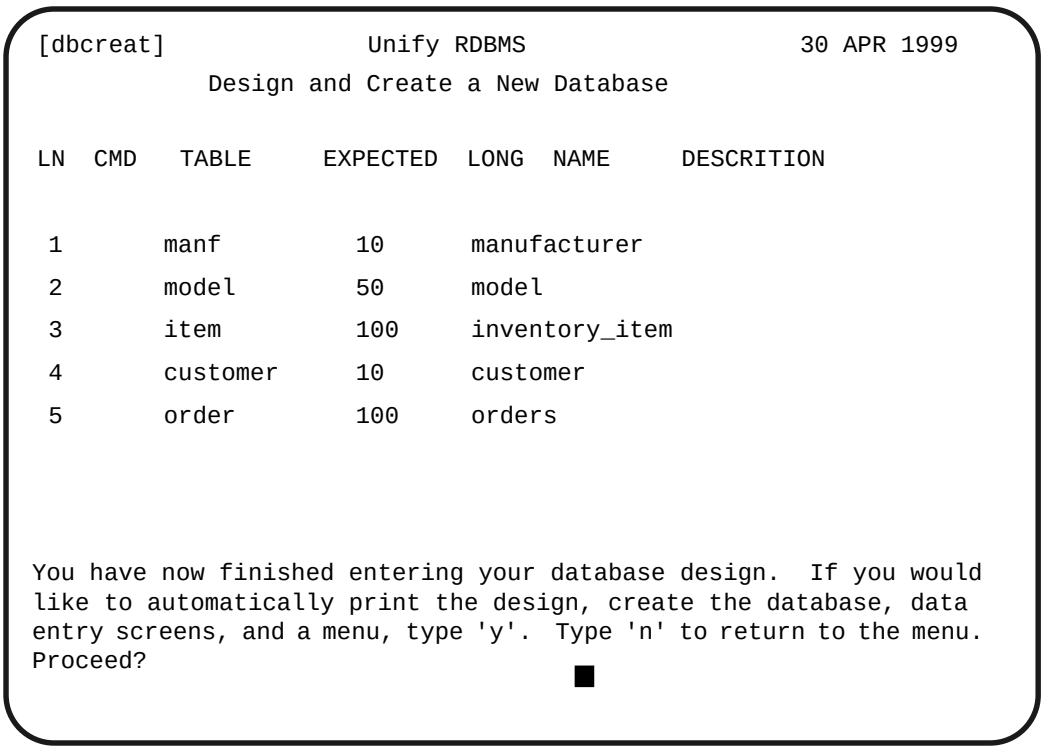
A combination field is not accessed by its long name but by its component field long names, as described above.

To add new fields to the current table, enter an a at the operation prompt to select add mode. The prompt clears and the cursor moves to the data entry area directly under the FIELD column heading. As in the table entry screen, RETURN moves the cursor forward, while CTRL U moves it backward.

Creating the database

The second phase of Design and Create a New Database lets you choose to print the database design, create an empty database, create data entry screens, and create a menu. These operations can be performed in several combinations.

When you exit the design phase with a CTRL U at the paging prompt, the following message displays: .



An n (no) answer ends Design and Create a New Database and redisplays the active menu. A y (yes) answer causes the following prompts to display:

```
Print  Database Design?     [  ]■ Create Database?  [  ]
Create Data Entry Screens?   [  ]   Create Menu?       [  ]

Type 'y' to select an option.  'n' to ignore one.
```

Move the cursor with RETURN to each prompt and enter either a y or an n to select your operations. You must answer each prompt to continue the process.

After you enter your selections, you are asked to verify the entries:

Are these choices ok?

The selection prompts are described as follows:

Print Database Design?

This option lists the tables, fields, and their relationships, as well as an alphabetical cross reference of fields and database size parameters. The report is 79 characters wide. See Print Database Design, section 11.1, for an output example. The following message displays:

The database design is being formatted for printing.

Create Database?

This option creates an empty database file from the design in the data dictionary. If you enter n (no) for this option, the last two selection options are skipped.

Because an empty database file is created each time you select the Create Database? option, you should take these precautions:

1. Make sure that no one else is using the application before starting.
2. The database should not contain valuable data, because the database will be emptied. If you have valuable data, dump it to an ASCII file using SQL. Create the empty database, and load the data back with SQL (section 16.3). It may be best to use

Reconfigure Database if you have valuable data in your database.

3. Raw databases cannot be created on a mounted file system. For more information about raw databases, see chapter 7, Introduction to Database Design and Access Methods, and section 9.5, Define Database Volumes.

   Before an empty database is created, you are again asked to verify that you really want to create an empty database:

   A new, empty database is about to be created. If you have information in an existing database, it will be lost. No one else should be using the database at this time. PROCEED?

After the database has been created, the screen looks like this:

```
  **Phase I   **

  **Phase II  **

  **Phase III **

The database file has been created. Hit RETURN to continue ->->
```

Create Data Entry Screens?

This option creates, processes, and registers with ENTER a data entry screen form for each table. The unique eight- character data dictionary name is used as the name of the screen. All existing screen forms with the same names as these tables are first deleted.

The screen heading is Table Long Name followed by "Maintenance."

The screen prompt for each field in the table is taken from the field's access name, usually the field's long name. If there is no long name, the field's implicit data dictionary name is used as the access name.

The screen prompt for a field that references a COMB type field is a combination of the referencing field's short database name and the short name of the referenced component field. For more information,

see this sections description of the LONG NAME column on the field entry screen.

Generally, the fields display on the finished screen in the order they appear in the table, from top to bottom. The prompts and fields are lined up one underneath the other in one, two, or three columns. The fields start after the longest prompt. Any underscore characters are changed to spaces, and appropriate letters are capitalized.

You can modify a default screen form using Check Screen Form Coordinates or Paint Screen Forms. You can also change the heading by using Register Screen Form with ENTER.

If you choose not to use this option, you can create screen forms for the database tables individually with Create Default Screen Form, Check Screen Form Coordinates and Paint Screen Forms. For more information, see the named sections.

This message appears for every table:

Creating data entry screen for: xxxx

where the variable xxxx is the name of the current table for which a default screen form is being made.

Create Menu?

dbcreate only allows this option if the Create Data Entry Screens? if you select this option. This option creates a menu titled "Data Entry Screens," with the name usermenu, and adds it as the last menu line of the Unify DataServer/ELS Main Menu. This menu contains a line for each of the default screen forms. The Data Entry Screens menu is recreated each time you select the option. Therefore, any changes made to usermenu using Add, Modify or Delete Menus are lost.

This message appears for every table:

Creating data entry screen and menu entry for: xxxx

where the variable xxxx is the name of the current table for which a default screen and menu entry is being made.

8.2 Modify database design (schent)

This chapter describes the use of the interactive database design option. You can use this option either to enter an entirely new design, or to modify the design of an existing Unify DataServer/ELS database.

The database design is stored in the Unify DataServer/ELS data dictionary. This data dictionary is a Unify DataServer/ELS database file that contains the system menus, screen forms, programs, users, and access privileges.

Changes to the data dictionary are made through the Table Entry and Field Entry screen forms of Modify Database Design (schent). These are the same two screen forms used by Design and Create a New Database.

The Table Entry screen form accepts general information about the tables in the database design. The Field Entry screen form accepts field definitions for each of the tables. The Table Entry and Field Entry screen forms are only a work area. Any changes made using these screen forms will not be seen in the application until Reconfigure Database is run. For more information on Reconfigure Database, see section 8.4.

Before making any changes to the database design;

1. Make a backup of your database.

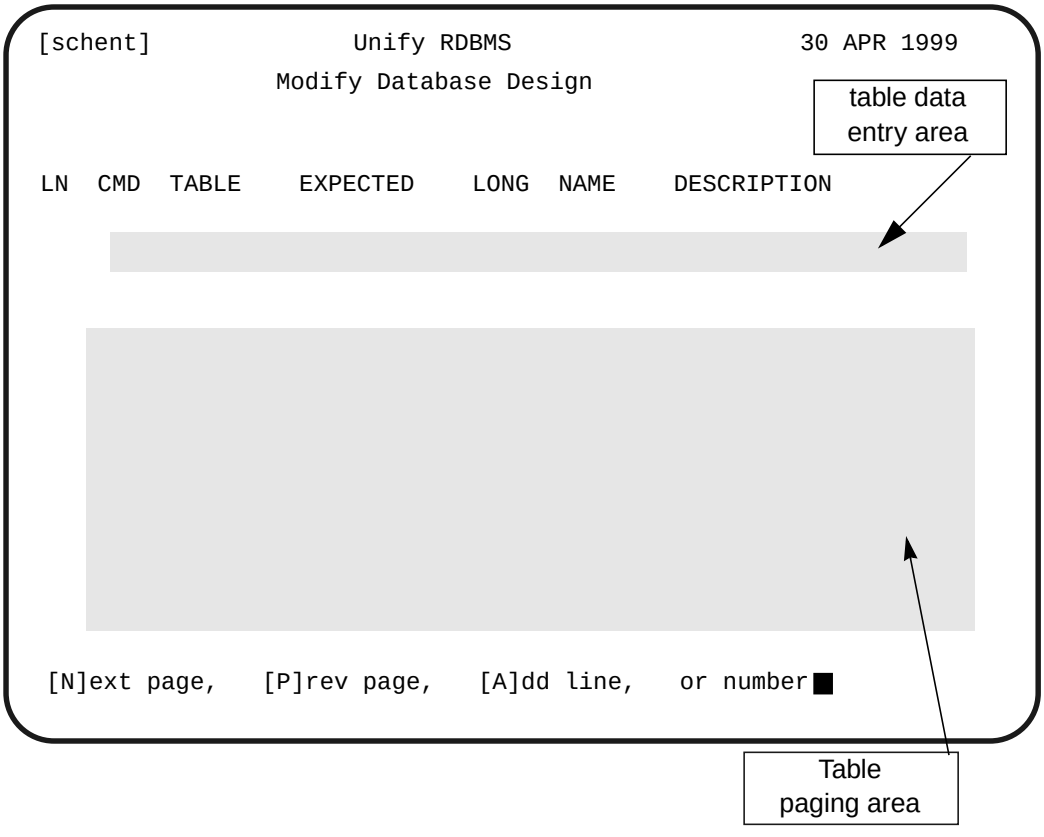
   A backup is needed in case you want to restore the original database after modifying the database design. See Write Database Backup, section 9.1.

2. Print a database design report.

   This report is useful to keep as a record of the database design before making any changes. See Print Database Design, section 11.1.

Modifying values on the table entry form

To use the interactive database design option, select Modify Database Design (schent) from the Database Design Utilities menu. The Table Entry screen form appears as follows:



The Table Entry screen form lets you add, modify, and delete tables in the database design. At the top of this screen form is a *table data entry* area where you can add entries. Below the data entry area is a *paging area* where you can modify existing entries.

Existing entries display in a page format, with 11 table entries per page. This is the same screen form used in Design and Create a New Database. For an explanation of the prompts, see section 8.1.

The following are column headings for the data entry area:

The remainder of this subsection describes each column heading. It also describes the steps you must follow to change, add, or delete data in the associated column.

TABLE

If you want to...	Then...
Change a Table's Name	1. If no data has been entered in this table, skip to step 2. If data has been entered to this table, dump data to an external file. See Dumping Data to External Files, in section 16.3 2. Delete the table definition. 3. Add the new table definition. You can use the table information in the Data Dictionary Report to re-enter the original table and field definitions. 4. Run Reconfigure Database (scom). 5. If no data has been entered in this table, you are finished. If data has been entered to this table, reload the external file data In the new table. See Loading Data From External Files, in section 16.3, or Database Load, in section 12.6.
Add a Table	1. At the Table Entry screen form, use add mode to enter the new table definition. If a Table Exists, Table can't have field names Field Exists, or Fields can't have table names message appears: a. Choose another name, or b. If you recently deleted a table or field name you want to use, run Reconfigure Database (scom), and try again. 2. At the Field Entry screen form enter the new field definitions. 3. Run Reconfigure Database (scom).
Delete a Table	1. At the Table Entry screen form, use delete mode to delete the table entry. 2. Run Reconfigure Database (scom).

Figure 8.5 Allowable Modifications to TABLE Entry

EXPECTED

Changing the expected record count affects the size of the hash table. Consequently, it also affects the size of the application database file, file.db. This can impact disk

space. However, making the expected number less makes the hash table and file.db smaller. This also decreases the number of record entries you can store.

If you want to...	Then...
Change the expected number of records	1. At the Table Entry screen form, use modify mode to access and change the Expected Number. 2. Run Reconfigure Database (scom). Answer y (or Y) to Rebuild Hash Index? prompt

Figure 8.6 Allowable modifications to EXPECTED entry

LONG NAME

Long Name is the name Unify DataServer/ELS uses to refer to a table.

If you want to...	Then...
Change a Table's Long Name	1. At the Table Entry screen form, use modify mode to access and change the Long Name. 2. Run Reconfigure Database (scom).

Figure 8.7 Allowable modifications to LONG NAME entry

DESCRIPTION

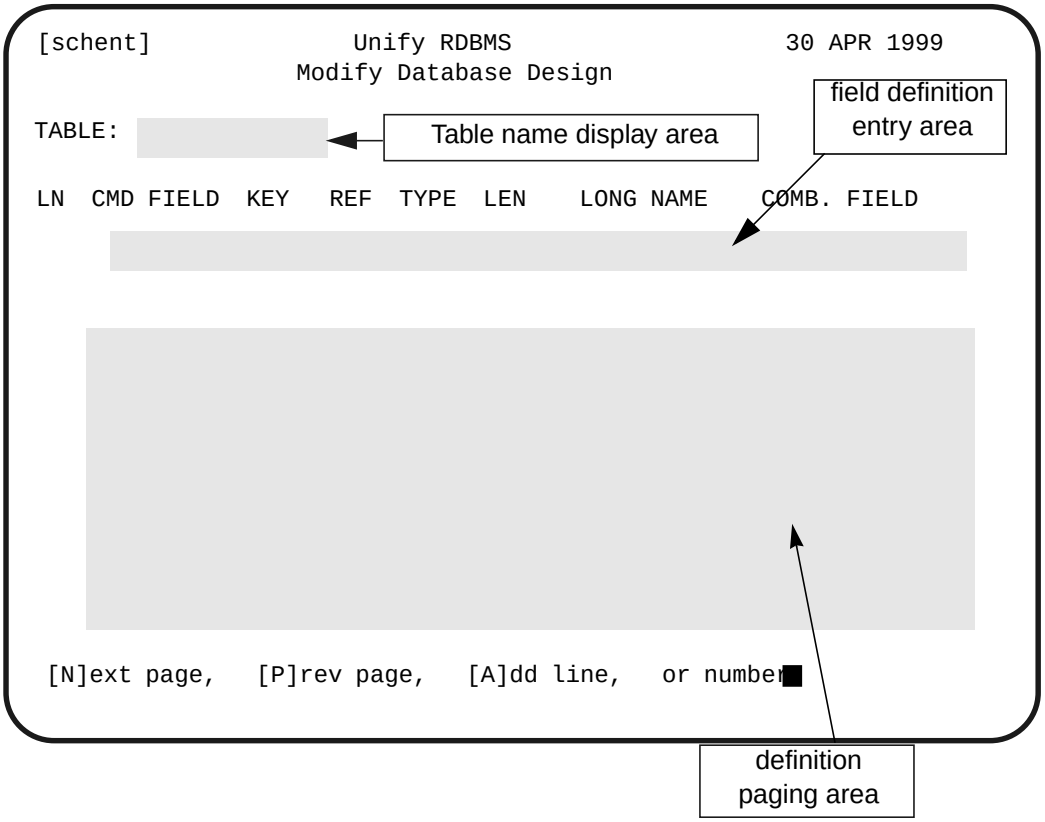
Description is used as a comment field.

If you want to...	Then...
Change a Table's Description	1. At the Table Entry screen form, use modify mode ato access and change the Description. 2. Run Reconfigure Database (scom)

Figure 8.7 Allowable modifications to DESCRIPTION entry

Modifying values on the field entry form

The Field Entry screen form is used to enter and modify the field definitions. Fields for tables may be added, modified, and deleted on this screen form:



The Field Entry screen form is similar to the Table Entry screen form. Both screen forms contain a data entry area and a paging area. Because the Field Entry screen form is also used in Design and Create a New Database, refer to Database Fields in section 8.1 for an explanation of the prompts.

FIELD If you want to change an existing key field or a COMB key component, see the KEY description in this subsection. See the KEY description if you want to change an existing non-key field to a key field or COMB key component.

Deleting a field deletes existing data in the field.

If you want to...	Then...
Change a Field's Name	1. If no data has been entered in this table, skip to step 2. If data has been entered to this table, dump data to an external file. See Dumping Data to External Files, in section 16.3. 2. Delete the field definition. 3. Add the new field definition. You can use the field information in the Data Dictionary Report to reenter the original field definitions. 4. Run Reconfigure Database (scom). 5. If the changed field is a key or COMB key component. Run Rebuild the Hash Table (rekey). 6. If no data has been entered to this table, you are finished. If data has been entered to this table, reload the external file data in the new table. See Loading Data From External Files, in section 16.3, or Database Load, in section 12.6.
Add a Field	1. At the Field Entry screen form, use add mode to enter the new field definition. If a Field Exists or a Fields can't have table names message appears: a. Choose another name, or b. If you recently deleted a table or field name you want to use, run Reconfigure Database (scom), and try again. 2. Run Reconfigure Database (scom). 3. If the added field is a key or COMB key component, Run Rebuild the Hash Table (rekey)
Delete a Field	1. At the Field Entry screen form, use add mode to delete the field definition. 2. Run Reconfigure Database (scom). 3. If the deleted field is a key or COMB key component, Run Rebuild the Hash Table (rekey)

Figure 8.8 Allowable modifications to FIELD entry

KEY — A primary key is marked by the key indicator (*) in the KEY column. A primary key that contains more than one field is a combination key. A combination key begins with the field type COMB (marked with the key indicator), followed by its component fields.

Deleting a field deletes existing data in the field.

Figure 8.9 Allowable Modifications to KEY Entry

If you want to...	Then...
Change an existing Key Field to a Non-Key Field	1. At the Field Entry screen form, use modify mode to remove the key indicator (*) from the Key column. 2. Run Reconfigure Database (scom). 3. Run Rebuild the Hash Table (rekey).
Change an Existing Non-Key Field to a Key Field	1. At the Field Entry screen form, use modify mode to add the key indicator (*) to the Key column. Note: A table can only have one key indicator before adding a new one. 2. Run Reconfigure Database (scom) 3. Run Rebuild the Hash Table (rekey).
Add a Key Field	1. See Add a Field in the Field section. Be sure to place a key indicator (*) in the Key column.
Delete a Key Field	1. At the Field Entry screen form, use modify mode to remove a field reference from the REF column. 2. See Delete a Field in the Field section.

Figure 8.9 Allowable modifications to key entry

REF A reference is used to relate a field of one table to the key of another table.

If you want to...	Then...
Add a Reference	1. At the Field Entry screen form, use modify mode to add a field reference to the REF column. 2. Run Reconfigure Database (scom). 3. Refer to Rebuilding Explicit Relationships, in section 9.4.
Delete a Reference	1. At the Field Entry screen form, use modify mode to remove a field reference from the REF column. 2. Run Reconfigure Database (scom)

Figure 8.10 Allowable modifications to REF entry

TYPE A Field Data Type describes how data for a particular field is displayed. Type also defines how the field is stored internally.

The valid data types are: AMOUNT, FLOAT, NUMERIC, DATE, LDATE, TIME, STRING, TEXT, BINARY, and COMB. For more information on Database Field Types, see chapter 7, Introduction to Database Design.

Changing a field data type can damage your data. Before making any changes to field data type, read Field and Internal Data Types in Appendix A. Figure 8.11 Allowable Modifications to TYPE Entry

If you want to...	Then...
Change a Field's Data Type	1. At the Field Entry screen form, use modify mode to change the data type in the TYPE column. If this field is a data type COMB, you must first remove the component field relationships to the COMB field or delete the component fields altogether. To change a COMB field relationship, see the COMB. FIELD section. 2. Run Reconfigure Database (scom). 3. If this is a key or COMB key compnent, run REBUILD the Hash Table (rekey).

Figure 8.11 Allowable modification to TYPE entry

LENGTH LENGTH controls the display length and internal storage size for the field.

You can change the display lengths for the following types of fields: NUMERIC, AMOUNT, FLOAT, STRING, and TEXT.

Changing a display length for a field can damage your data. Before making any changes to field lengths, read Field and Internal Data Types in Appendix A.

If you want to...	Then...
Change a Field's Display Length	1. At the Field Entry screen form, use modify mode to change the filed's display length in the LEN column. 2. Run Reconfigure Database (scom) 3. If this is a key or COMB key component, run Rebuild the Hash Table (rekey).

Figure 8.12 Allowable Modifications to LENGTH Entry

LONG NAME The field long name is the field name Unify DataServer/ELS uses to refer to a field.

If you want to...	Then...
Change a Field's Long Name	1. At the Field Entry screen form, use modify mode to change the filed's long name in the Long Name column. Be sure that the new name is unique. 2. Run Reconfigure Database (scom).

Figure 8.13 Allowable modifications to LONG NAME entry

COMB. FIELD

You can add or remove a COMB component field relationship in the COMB. FIELD column.

If you want to...	Then...
Add or Remove a COMB Component Relationship	1. At the Field Entry screen form, use modify mode to access the COMB. FIELD column. 2. To add the field to an existing COMB Field, enter the COMB field name into the COMB.FIELD column. To remove a COMB component field, enter spaces (with the space bar) to the COMB.FIELD column. 3. Run Reconfigure Database (scom). 4. If this is a key or COMB key component, run Rebuild the Hash Table (rekey).

Figure 8.14 Allowable modifications to COMB. FIELD entry

8.3 Create database (crdb)

This program is used to create an initial database file from the database design in the data dictionary. Create Database is most commonly used to create a database that has been designed with Modify Database Design.

The general process consists of compiling the database design, determining the type of file the database is to reside in, and then writing the initial file. These steps are done automatically by the program. Because Create Database creates an empty database file, certain precautions should be observed:

1. Make sure that no one else is using the application before starting. No one else should be using this Unify DataServer/ELS application when Create Database is running.
2. The database should not contain valuable data, because the database is empty when Create Database finishes. If you have valuable data, dump it to an ASCII file using SQL, and then load it back (see SQL Extensions in section 16.3) when the create program is finished.
3. Create Database also empties all existing B-tree indexes. If the data types or internal lengths of existing fields have been modified since the last Create Database run, any B-tree indexes associated with those fields are dropped.
4. Raw databases cannot be created on a mounted file system.

Database files

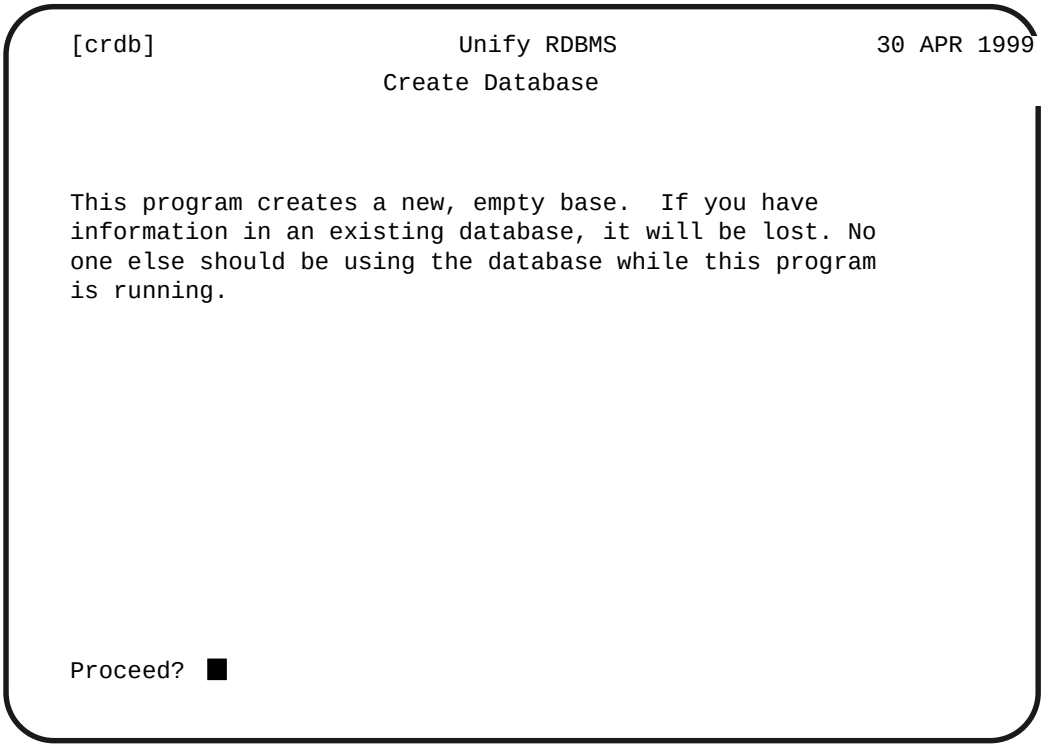
The default Unify DataServer/ELS database file is an ordinary operating system file named file.db in the current directory.

A Unify DataServer/ELS database file can reside in either an ordinary or a special operating system file. To define the type of file, set the appropriate parameters with Define database volumes (section 9.5).

If a special file is used, file. dbr is set to point at the raw device that the data file is on. Therefore, file.dbr has meaning only if a special file is being used.

If the expected size of the data file exceeds one megabyte, creating the database as a special operating system file can increase system speed and efficiency significantly. This allows programs to use the raw device for input and output, and bypass the operating system file indexing and buffering database designs. Refer to Define Database Volumes, section 9.5, to see how to create special files.

To create an empty database, select "Create Database" (crdb) from the Database Design Utilities menu. The Create Database (crdb) screen form displays:



Note that if a database exists, it is emptied so that it contains no records.

Answer Y to start the database creation process, N or CTRL U to stop and return to the Database Design Utilities menu. The program displays the messages shown on the following screen as the process continues.

```
[crdb]                  Unify RDBMS                  30 APR 1999
                     Create Database


 **Phase I   **

 **Phase II  **

 *Phase III **









  Process complete.  Back up the data ->->
```

Create Database creates the database as an ordinary operating system file in the current directory. After the process ends, press RETURN at the final prompt, to redisplay the system menu.

8.4 Reconfigure database (scom)

A Unify DataServer/ELS database must be reconfigured after you make changes to the database design. Changes requiring the database to be reconfigured include:

- Adding, Deleting or Modifying table definitions
- Adding, Deleting or Modifying field definitions

For information about changing the database design, see Modify Database Design (schent), section 8.2. For information about creating a new database, see Create Database (dbereate), section 8.3.

When you create or modify tables and fields, the changes are recorded in the data dictionary file, unify, db. Reconfigure Database then uses the data dictionary information to create or reformat your application database, located in file.db.

NOTE: If you update a data dictionary that you use on several different machines and you want to apply the updated data dictionary to the other databases, copy the updated data dictionary (unify, db) to the other machines before reconfiguring. Reconfigure each copy of the updated data dictionary separately, to avoid database-data dictionary mismatches.

You can also use Reconfigure Database to restore file system or volume information to file.db. You must do this when you move an existing database to raw files. For more information about database volumes, see Define Database Volumes, section 9.5.

The reconfigure process has four steps:

1. Reconfigure locks out all database updates.
2. Reconfigure copies the data dictionary. Then, if a fatal error occurs, such as a disk or tape read/write error, the dictionary can be reset to its original state.
3. Reconfigure compiles the database design and generates tables to translate records from the old format to the new.
4. Reconfigure writes the database to a temporary file, then reads the database back in its new format.

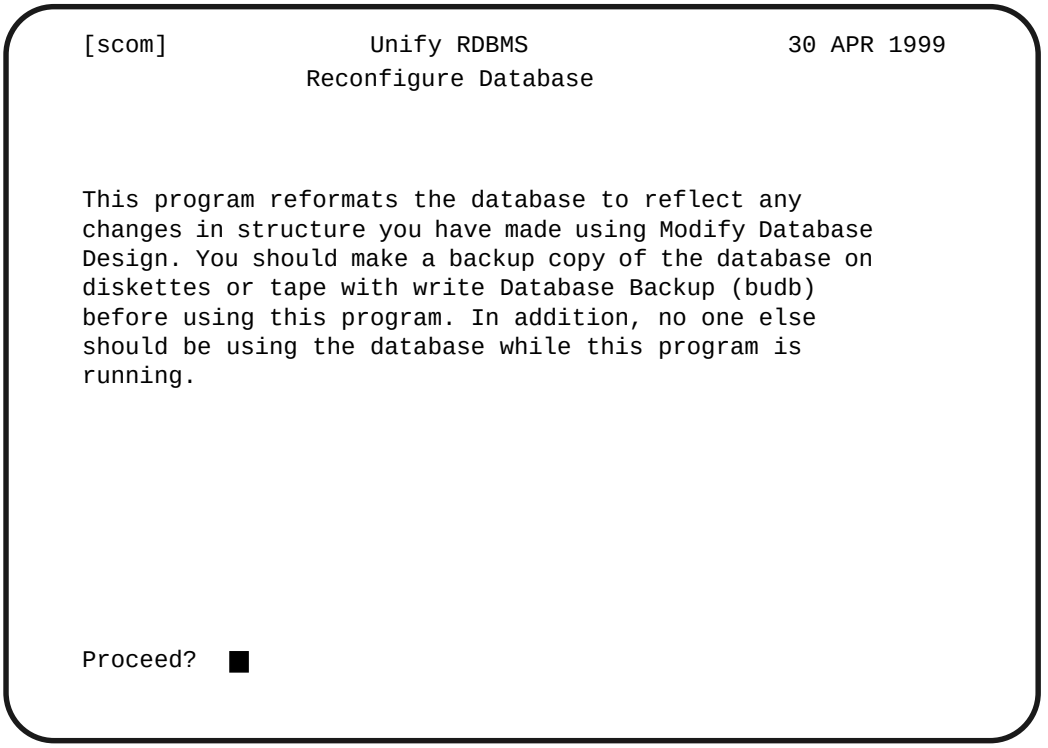
For reconfigure error messages, see the end of this chapter.

Preparing to reconfigure

1. Make sure no one else uses this database during the reconfigure process. This includes querying, updating, or adding records to the database, or changing the data dictionary.
2. Back up the database. See section 9.1, Write Database Backup.
3. If the database file is too large to reconfigure on the current disk, Reconfigure Database will give you the option of using a diskette or tape as a temporary work area. Reconfigure Database uses the device specified by the BUDEV environment variable. If you use this option, make sure the BUDEV environment variable is set. For more information about setting the BUDEV environment variable, see chapter 5.
4. Reconfigure limits the actual number of fields and relationships to 300 per table. If this number is exceeded, Reconfigure stops processing and displays an error message.

Reconfiguring

Select Reconfigure Database (scom) from the Database Design Utilities menu. The following screen form displays:



Enter y (or Y) at the PROCEED? prompt to start the reconfigure process. An n (or N) or a CTRL U stops the process and returns to the menu.

If you need to rebuild the database hash table, the following messages also display:

```
[scom]              Unify RDBMS                  30 APR 1999
                 Reconfigue Database



**Phase I   **

**Phase II  **


Nominal hash table loading factor:   50%
New load factor without rebuilding:   nn%









REBUILD HASH INDEX?
```

If you changed the expected number of records for any table in the database, you should rebuild the hash table. The hash table is rebuilt using Rebuild the Hash Table (rckey). Answer y (or Y) at the prompt to rebuild the hash table. Answer n (or N) to leave the hash table as is.

Reconfigure continues with the following display and prompt:

```
[scom]              Unify RDBMS                  30 APR 1999
                  Reconfigure Database



 **Phase I   **

 **Phase II  **

 **Phase III **









 Use diskette/tape as the temporary file?
```

This prompt gives you the option of using a diskette or tape as a temporary work area. Answer y (or Y) to use a diskette or tape for the temporary work area. Reconfigure Database uses the device specified by the BUDEV environment variable. Answer n (or N) to reconfigure on the current disk.

If your database is very large, or you have little free file space, you may not be able to reconfigure without using off-line storage. The temporary files created by Reconfigure require as much storage space as unify.db and file.db combined.

Finishing the reconfigure

1. If you have changed the internal data type of any key field, you must run Rebuild the Hash Table (rekey). Otherwise, you will not be able to access the affected table by its primary key.

NOTE: If Reconfigure has rebuilt the hash table, skip this step. (See REBUILD HASH INDEX? prompt description.)

2. If you have changed an existing ordinary field to an explicit relationship, Reconfigure does not build the necessary pointer lists. To rebuild pointer lists, you must run Rebuild explicit relationships (repoint). See section 9.4.

3. Reconfigure does not update existing screen forms with changes to the database. To update or delete existing screen forms, you must use Paint screen forms (paint) or ChEck Screen Form Coordinates (sfmaint). Use Create Default Screen Form (cdsf) to create a default screen form. See sections 13.1, 13.2, and 14.1.

   Often screens do not need to be modified, but they do need to be reprocessed. To reprocess screens, use Compile screens (sfproc), with the all option. See section 14.4.

4. Reconfigure drops any B-tree indexes and definitions associated with fields that have been deleted or fields whose internal data type or length has been changed. See Add, drop B-tree Indexes, section 8.5

5. Custom programs that access modified tables and fields must be recompiled and reloaded. If you are using Advanced field attributes, you must recompile the afa source file (field, afa).

6. Reconfigure turns off transaction logging. To restart transaction logging, back up the database with Write database backup. See section 9.1.

7. Verify that the changes to the database are correct. If the new design and resulting data have converted correctly, back up the new database with Write database backup. See section 9.1.

   If you want to restore the original database, read in the original database backup you made with Read database backup. See section 9.2.

Recovering corrupted hash table fields

The dbnode record of the data dictionary (unify.db) can become corrupted if crdb is started but is aborted before completion. For example, if crdb is started on one terminal but then killed from another after it had begun processing, the unify.db file can be left in an inconsistent state.

If this has occurred scom displays the following message after the "Proceed?" prompt:

```
An internal error has occurred. Please check the error log.
```

This message creates the following error log entry in the error log file, erriog:

```
Program: scom
Calling function: error
Offending function: See notes.
      Status: 0
Errno: 0
Notes: The dbnode has table field in unify.db are corrupted.
Please report this error to Unify
           Customer Support.
```

To correct this problem, you need to check the value of the hash_length field in the dbnode table of the Unify DataServer/ELS data dictionary, unify.db.

If it has become corrupted, the hash_table field is zero (0) when the hash_start and hash_end fields indicate that the hash table should have a nonzero length.

To check the value of hash_length, perform the following steps:

1. Restore your database from a backup.
2. Set the DBNAME environment variable to unify.db:

   DBNAME=unify.db

   export DBNAME
3. Start up SQL. SQL
4. At the SQL prompt, get the values of the fields hash_start, hash_length, and hash_end from the table dbnode.

   select hash_start, hast_length, hash_end from dbnode /

   A sample output is:

```
hash_start  |  hash_length   |    hash_end
-----------------------------------------
  41984     |         0      |      692264
```

5. Compute the correct value for the hash_length field using the following formula:

   (hash_end - hash_start) / 3 = hash_length

   For example, using the previous sample output, the correct hash_length value should be:

   (692264 - 41984) / 3 = 650280 / 3 = 216760

6. At the SQL prompt, update the hash_length field to its correct value.

   update dbnode
   set hash_length = value /

   where value is the correct hash_length as calculated according to the formula.

7. Verify that the correct hash_length value now exists in the dbnode table.

   select hash_start, hash_length, hash_end from dbnode/

   The sample output should now be:

```
   hash_start  | hash_length  |   hash_end
-----------------------------------------
       41984   |      216760  |    692264
```

8. Exit SQL.

   end /

   Do not remove the unify, db file.

9. Run the reconfiguration program, scom, again.

Since the hash_length value should no longer be zero, the unify.db file should no longer be corrupted. However, if you find that scom sets the hash_length back to zero, please call Unify DataServer/ELS Customer Support.

8.5 Add, drop B-tree indexes (idxmnt)

A B-tree is an access method that uses a binary search method to find the rows that match a field value. B-tree indexes speed up random and sequential retrievals on large databases. However, B-tree indexes make updates slower and use more disk space. In general, you should specify B-tree indexes on fields that you use frequently in queries.

Although you could specify a B-tree index on every field in a table, it is not recommended. This is because of disk space requirements and additional overhead imposed when updating the fields.

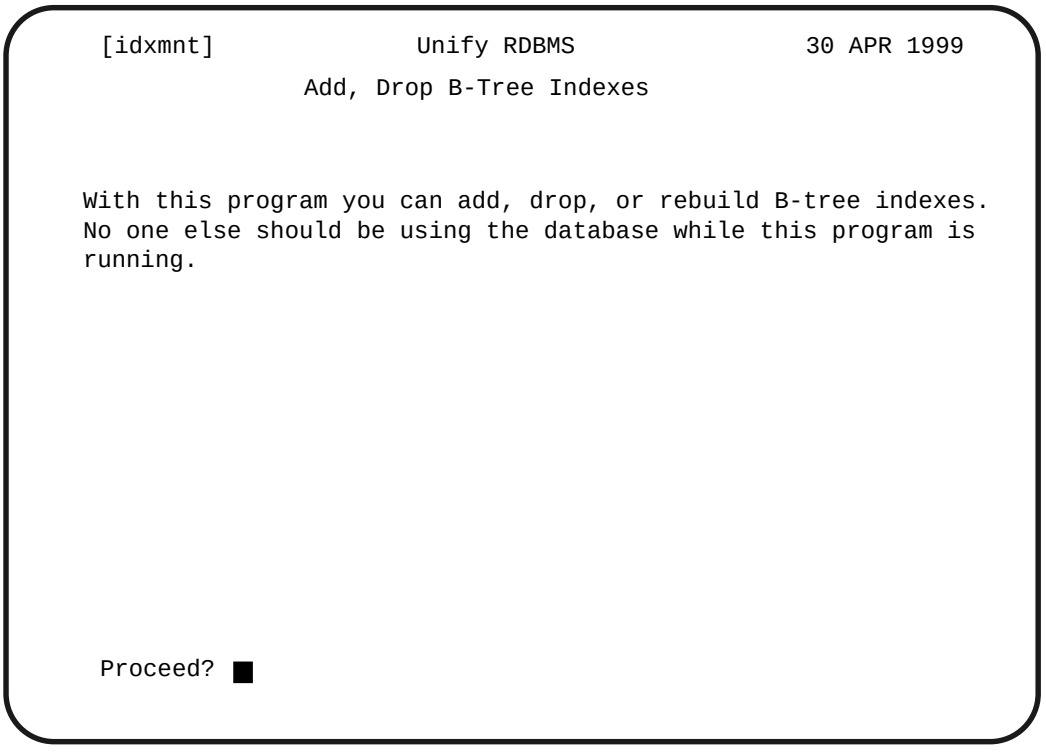
Up to eight fields can be placed in a single B-tree index, and each field can be stored in either ascending or descending order. All the fields must be contained in a single table.

You might put more than one field in a B-tree index if you frequently use two or more fields in queries. Note that you can also specify that a B-tree index contain only unique values. B-tree index structure is detailed in the *Unify DataServer/ELS Direct HLI Programmer's Manual*.

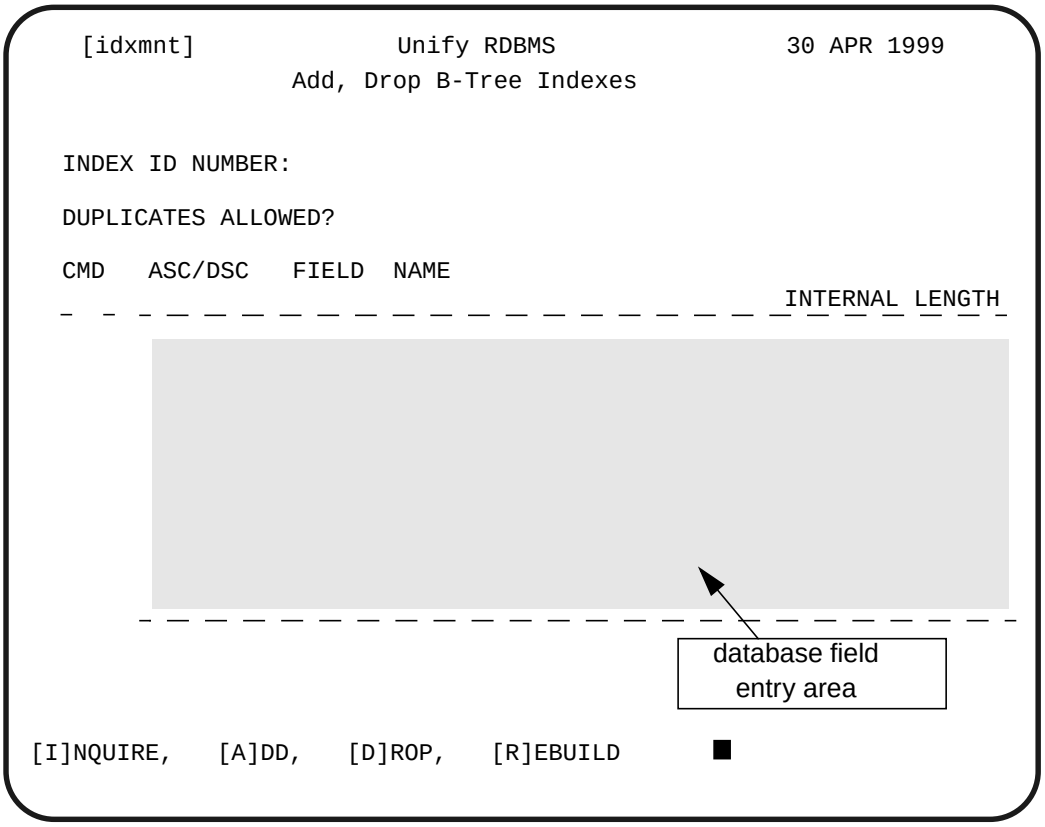
B-trees must resde in SDBPATH. However, if your operating system supports symbolic links, you may set up symbolic links to a raw partition, to a different file system, or to a different directory.

Before you use Add, drop B-tree Indexes, make sure no one else is using the database. To add or rebuild a B-tree index, or to drop an existing index, select "Add,

drop B-tree Indexes" (idxmnt) from either the Database maintenance menu or Database design utilities menu. The following screen form displays:



If you answer y (yes), the next screen form appears; a n (no) terminates the program.



When the program starts, it checks the database directory for necessary B-tree index files. If any files are missing, you are asked to rebuild them. The rebuild process is included in the following description.

After the index file check, the operation prompt displays:

[I]NQUIRE, [A]DD, [D]ROP, [R]EBUILD

The valid operation selections are described as follows:

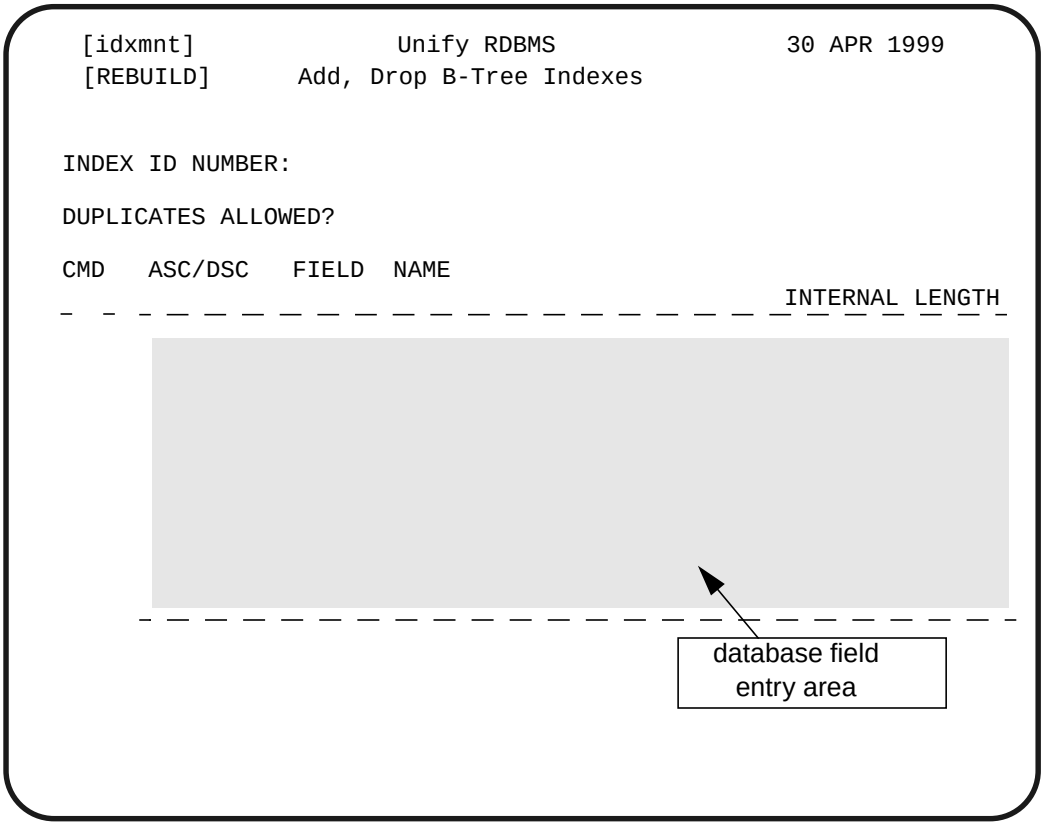
i	Inquire about B-tree index definitions
a	Add B-tree index definitions
d	Drop B-tree indexes

r Rebuild B-tree indexes. Rebuild the indexes when:

- The system asks you to do so
- There is a problem with the B-tree index
- You want to reduce the size of the B-tree index file after you delete a large number of records in a table

Inquiring, rebuilding, and dropping B-trees

When you select inquire, rebuild, or drop mode, the screen displays the current mode in the top left corner. For example, in rebuild mode, the screen displays as follows:



To select a B-tree to rebuild, you can either enter a B-tree ID number at the INDEX ID NUMBER: prompt or a table name at the TABLE: prompt. In add mode, both prompts are display-only. These two prompts are explained as follows:

INDEX ID NUMBER:

The B-tree index identification number. This ID number is assigned by Unify DataServer/ELS when an index is added, to identify the index until it is deleted. Index identification numbers are from 1 to 255.

You can only search and select on one ID number at a time.

TABLE:

The name of the table associated with the B-tree index. This prompt does not display in add mode until the first database field has been entered. All fields listed in the current B-tree index definition must be in this table. Search and selection can be performed for all indexes associated with a single table.

When a table with more than one index is selected, the following prompt displays:

[N]ext index, [P]revious index, [D]rop, [S]top

This paging prompt lets you step through the B-tree index definitions selected in inquire, drop, and rebuild modes.

n	List the next index definition. RETURN also advances to the next definition.
p	List the previous index definition
d	Drop the current index definition. This option displays only in delete mode.
s	Stop the search/selection display mode

Rebuilding B-trees is not necessary after a database reconfiguration. However, if you have deleted records from a B-tree, you can rebuild the B-tree so as to reduce its size and improve performance.

Adding B-tree indexes

When you add a new B-tree index, you must respond to the following prompt:

DUPLICATES ALLOWED?

If you answer N (no) at this prompt, you set a flag indicating that fields listed in the current table's index definition cannot have duplicate values.

All fields making up a B-tree index are a set. The no-duplicate condition is tested on those field values as a set. Therefore, all field values of that set must match another record's value set before the new record is rejected as a duplicate.

A CTRL U at the DUPLICATES ALLOWED? prompt ends the mode and adds the current B-tree index.

NOTE: Default values of any field type are duplicates just like other values. Therefore, if you don't want a B-tree index to have duplicates, at least one field in the index set must be given unique values. If one field in the set is always unique, the set is unique.

The middle section of the screen form is where you enter the database fields to be included in the current B-tree index. Remember, fields used in a B-tree index definition must come from the same table.

Database field entry prompts

The following describes the prompts in the Database field entry section:

CMD The column where you enter an operation command for the current line. The position of the cursor marks the current line. The cursor moves up or down in this column when you press CTRL U or RETURN. Valid commands in this column are follows:

- a — Add a database field name to the current index definition
- d — Delete the current field name from the definition

q — Quit add mode and return to the DUPLICATES ALLOWED? prompt

ASC/DSC

a — Specifies that the field values are to be stored in the index in ascending order

d — Specifies that the field values are to be stored in the index in descending order

You must enter a or d for each field specified in the Database Field Entry section.

FIELD NAME

The name of an existing database field that is (or will be) in the current B-tree index definition. The field cannot be a COMB, TEXT, or BINARY field. All field names must be in the same table.

Any field used in a B-tree definition is indexed as defined.

INTERNAL LENGTH

A display-only field. This value describes the length of the field's value when stored in the B-tree index. The system computes the internal length.

Processing B-tree definitions

When you add or rebuild a B-tree index definition, the following message displays below the second line:

```
THE SUM OF INTERNAL LENGTHS FOR ALL THE FIELDS CANNOT EXCEED nnn
```

The value nnn is determined by the configuration of the operating system on which Unify DataServer/ELS is running. You may have to reduce your index definition to fit in this size limit.

After you add an index or modify an existing one, you see the following prompt:

The B-tree index will now be created. Continue?

This means that the entries you have made in the index definition are valid.

If you are satisfied that the definition is correct, and you want the index to be created as defined, answer y (yes).

If you answer yes, the following message displays:

filling index

The program increases the record count by tens as it indexes records.

If you answer n (no), do not create the index, the following message displays instead:

Do you want to discard this B-tree index definition?

The program confirms that you want to delete this B-tree index definition. If you answer no to this prompt, you can then change the definition.

In rebuild mode, you can rebuild a single B-tree index or all the indexes in the database at one time. If you press RETURN after the first two prompts on the screen form you see the next prompt:

Do you want to rebuild ALL of the indexes?

If you answer y (yes), all the defined B-tree indexes are rebuilt. As each index is rebuilt, its definition displays. Once this process has begun it doesn't stop until all indexes are rebuilt. If you have many indexes defined, or a large database, this can take time.

8.6 Advanced field attributes (afa)

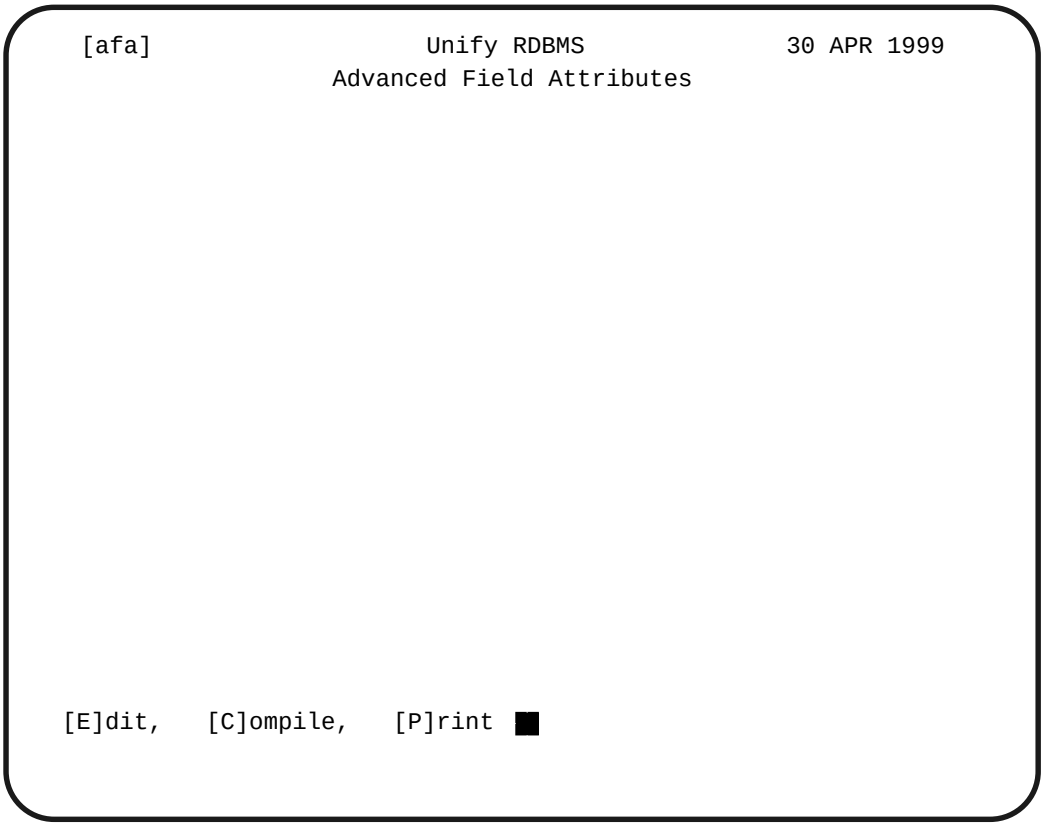
This chapter describes the Advanced Field Attributes you can define for each field in the database design. The basic attributes of a field are its length and data type such as STRING, NUMERIC, or DATE.

Advanced field attributes include a default value, a list of legal values, a one line information message, an error message, and the name of a help file with additional information on the field.

When you specify default or legal values for a field, ENTER and SQL use the values to initialize the fields in newly added records, and to validate proposed field updates. When you specify FYI (For Your Information) error and help messages, ENTER displays them when the user needs help or correction filling out the screen form. These attributes are also available from the Host Language Interface.

Unify DataServer/ELS has a menu option, Advanced field attributes, that enables you to create, edit, compile, and print an Advanced field attributes source file. To run the option, select "Advanced field attributes" (afa) from the Database design utilities menu.

The following screen form displays:



The only prompt on this screen form is the operation prompt:

[E]dit, [C]ompile, [P]rint

The valid responses to the operation prompt are:

e — Edit the Advanced field attributes source file, fields.afa. This command can be used to create or modify an attributes file. Each time you modify the attributes source file, you must compile it so changes can be used by ENTER and SQL. The default editor is UC Berkeley vi. You can change the editor by setting the EDIT environment variable.

c — Compile the Advanced field attributes source file, fields, afa, and send the output to the file, unify, afa. You must recompile the source file each time after a database reconfiguration.

If any errors occur during the compile process, error messages are sent to the file fields.afa.err. The program re-enters edit mode to let you view and modify both the error and source files.

p — Print the Advanced field attributes source file, fields.afa. Output is sent to the device specified by the SPOOLER environment variable or to the default print spooler device, lpr.

CTRL U — Exit the Advanced field attributes program; return to the current menu.

The Advanced field attributes, afa, compiler can also be run from the shell. Use the following command syntax:

afac *source-file error-file*

Where *source-file* is the name of the file containing the Advanced field attribute definitions. If you don't want error messages sent to the standard error device, you can specify a file for error messages with the optional *error-file*.

After you enter the afac command, the compiler creates the unify.afa file in the directory set by DBPATH, or in the current directory if DBPATH is not set.

The maximum file size of unify.afa is 64k bytes. However, machines with limited memory may not be able to accommodate a 64k-byte unify.afa file.

When Advanced field attributes are specified, all programs that interface with AFA must be able to read the unify.afa information into memory. Programs that interface with AFA include ENTER, SQL, and custom (HLI) programs. If the programs that interface with AFA cannot read the unify.afa information into memory, the Advanced Field Attributes cannot work, as if they were never created.

If any of the programs that interface with AFA cannot read unify.afa into memory, try one of the following solutions:

1. Define and compile fewer Advanced field attributes.
2. Decrease the amount of memory needed by the program.

AFA file structure

The general structure of the Advanced field attributes file is as follows:

```
table section
        field section
                statement
                statement
                .
         field section
                .
                .
                .
 table section
        .
        .
```

Each table section contains one or more field sections. Each field section contains one or more statements. Statements normally take a single line, but they can be continued on additional lines by putting a backslash (\) at the end of the line.

The following describes how to specify Advanced Field Attributes. This includes a syntax diagram for each section.

The syntax diagrams use the following conventions:

bold items These words or characters are required in the attribute statement or are optional keywords. The words have special meaning, and cannot be used for table or field names.

italic words These are place holders for words, numbers, or expressions you define. Examples include *table* for a table name, *field* for a field name, and *offset* for a constant value.

[ ] Square brackets around a word or expression means that the enclosed item is optional, and can be left out.

| | Vertical bars enclosing a stacked list of options mean to choose only one of the options.

Command syntax

RECORD-TYPE SECTION

record-type table
field section
...

The record-type statement is used to establish the current table. This is necessary because fields are not required to have unique names. You need only one table section for each database table. All the fields listed in a table section must belong to the current table.

table — The name of a table in the database design.

FIELD SECTION

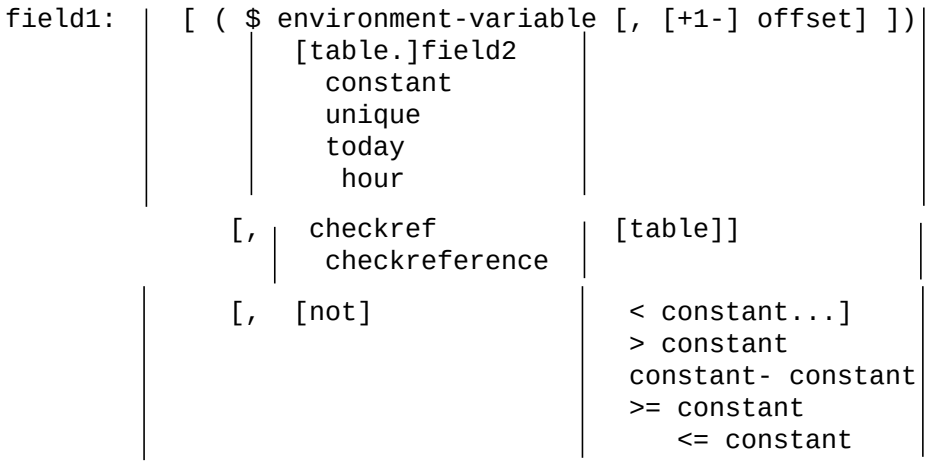
```
field1: [(default-value [, [+|-] offset])]

   [,  checkref       [table]]
       checkreference

   [, legal-values]

[ FYI=' string' ]
[ ERRORS string']
[ HELP= filename ]
```

A more complete definition of the field statement follows.

For STRING fields only:

```
field 1: [ ( $environment-variable ]] [, 'string'... ]
             [ table.]field2
                 'string' ,

         [,  checkref       [table]]
             checkreference
```

For NUMERIC, FLOAT, AMOUNT, DATE, LDATE, and TIME fields:



field 1 The access name, usually the field's long name, of a database field in the current table. This is the field whose attributes are being described. The field cannot be a COMB type.

environment-variable
The name of any environment variable. The variable must be set and its value exported.

table An optional name of the table that contains field2, if field2 is not in the current table.

field2 The access name, usually the long name, of a database field. If this name is not preceded by a table name, it is assumed to be in the current table. This lets you specify a default value from another database field.

NOTE: If. field! is from a different table, a record in that table must be current when this record is added. Otherwise, it won't be possible to get the field value. This is most useful when used with the Host Language Interface.

string A sequence of printable characters. Strings used as default values are stored literally in the field. You can use the single quote character in a string if you precede the quote character by a backslash character (\). This also applies to the backslash character itself.

When you define the legal values for a string field, you can use Regular Expression Notation, described under "Field Section Statements" in this section (8.6).

constant — A fixed value of type NUMERIC, DATE, LDATE, TIME, AMOUNT, or FLOAT. The data type of the constant must match the data type of the current field.

unique — This sets the default value of the field to the current record number. Use this to assign unique key values. You can only use this default for long NUMERIC fields (NUMERIC 5-9).

today — This sets the default value of a DATE or LDATE field to the current month, day, and year.

hour — This sets the default value of a TIME field to the current hour and minutes. The time is formatted as hh:mm.

offset — The amount the default value is reduced or increased before the value is stored in the field.

Fields can be offset in the amounts shown in Figure 8.15.

Data Type	Offset increment
NUMERIC	integer
DATE	day
LDATE	day
TIME	minute
AMOUNT	penny
FLOAT	integer
STRING	none

Figure 8.15 Field offsets

The table in Figure 8.16 contains examples that show how default values can be offset.

When offset is...	Then the value of field 1 is...
today+1	one day from the day the record is entered in the database table
hour +3	three minutes from the time the record is entered in the database table
unique+10000	a unique value starting with 1001, instead of 1
Sal_Amt+2.75 to a Key Field	the value of Sal_Amt in the current record, plus 2.75, where Sal_Amt is the name of a field in the current table

Figure 8.16 Default value offsets

checkreference

checkref

The checkreference or checkref clause enforces data integrity. It only allows a value to be entered for a field if the value matches an existing key field value in a referenced table.

Either the checkreference or checkref keyword can be used in the clause.

The data type and length of field] must be identical to the referenced key field's data type and length. You can use table to specify the table containing the key field to be referenced, if field1 does not have an explicit relationship with a key field defined in the database design.

The table in Figure 8.17 describes the uses of the checkreference clause.

When field1...	Then table...
has an explicit relationship with a key field defined in the database design	may be entered, or another table entered if its key field has the same data type and length
does not have an explicit relationship with a key field defined in the database design	must be entered

Figure 8.17 Uses of the checkreference clause

"An example file," later in this chapter, shows how to use the checkreference clause.

not — This keyword negates all the legal values that follow it. The value entered can be anything except the values listed.

filename — The name of an ASCII file that contains the help documentation for the current field. This file must be located in the ../hdoc directory, relative to the DBPATH environment variable or relative to the current directory if DBPATH is not set.

FIELD SECTION STATEMENTS

The field section consists of the following statements:

- field statement
- FYI message statement
- error message statement
- help statement

The Field statement

Use the *field statement* to set *default values* and *legal values* for a field. Enclose default values in parentheses, then list comma-separated legal values.

The default value is used when a new record is added to the database. This value is not checked against the legal values for validity. If you have not specified a default value for a field, then the default values shown in Figure 8.18 apply.

Data type	**Default**
STRING	null (blank displayed
TEXT	null
BINARY	null
NUMERIC	0
AMOUNT	0
DATE	**/**/**

Figure 8.18 Field default values

Data type	Default
LDATE	**/**/**
TIME	00:00
FLOAT	0

Figure 8.18 Field default values

If the field is NUMERIC, FLOAT, AMOUNT, DATE, LDATE, or TIME, when you specify a default value for a field, you can also include an offset. The offset is added to or subtracted from the default value before the value is stored in the field. See Figure 8.15 for the field offsets.

The following example illustrates why you may want to have an offset. Suppose your table has a field that has a unique ID number for every piece of equipment (equip_ID). You want all equipment ID numbers to be over 5000. You can have the system automatically assign this number by using the following field statement with a default and an offset value:

equip_ID: (unique ,5000)

This statement causes the default value for equip_ID to be set to the current record number, plus 5000.

Considerations

Problems can occur when values for a field with a unique default attribute definition are entered manually or are loaded into a database from another database.

1. The first problem occurs when you manually enter a value into a field that has a unique default definition. This can result in a future duplicate value being generated for the field. When this happens, no more records can be added to the database until the duplicate value is changed to a unique value.

   To avoid this problem, either let the system enter the value for a field with an AFA unique default value definition, or add an offset value to the AFA unique default value definition to reduce the chances of a duplicate

   For example, suppose you have a field named keyfid with a unique default value definition. You can add an offset of 10,000 to the unique value with the following statement:

   keyfld: (unique,+10000)

This means that all default values for keyfid would be unique numbers greater than 10,000. You would be able to manually enter unique values from 0 (zero) to 9,999.

2. The second problem occurs when you dump records from one database table and reload them into another database table with a unique default value definition for one or more of its fields. This can cause many duplicate field values.

   To avoid this problem, either only load records into tables without unique default value definitions, or modify the AFA definition so the unique field uses an offset greater than any of the unique field values of the reloaded records.

Legal values

You can also use the field statement to specify the legal values that can be entered for this field. If the value entered for a field matches any legal value set in this statement, then the value is accepted.

You must enclose legal values for STRING fields in single quotes (').

Regular expression notation

In addition to literal character strings, you can use a more powerful regular expression notation to set AFA legal values for STRING types. This is similar to the way you specify matches in ENTER, SQL, and RPT, but regular expression notation is more comprehensive.

The following special characters are recognized in the regular expression notation:

[ ] Character class. The *character class* matches any character that is a member of the class enclosed in the brackets.

^ A caret in brackets negates the character class. It must be the first character after the [.

For example, '[^abc]' recognizes any character except a, b, and c.

- A dash in brackets specifies a range of characters. It can be a member of the class only when it is the first

character (after the caret, if any) or last character of the class.

For example, '[0-9]' equals '[0123456789]'.

] A right bracket ends the character class. It can be a member of the class only when it is the first character (after the caret, if any) of the class.

For example, '[^]-]' matches any character except the right bracket and the dash; and '[]A-G]' matches the right bracket or A through G, inclusively.

. A period matches any character.

* An asterisk matches the previous regular expression zero or more times.

The following examples show how the period and asterisk (.*) are used.

The period and asterisk ('.*') match a string of 0 or more characters.

'John.*' matches any string starting with John, such as Johnny or John Smith.

'a*' matches null strings or the strings a, aa, or aaa.

+ A plus symbol matches the previous regular expression one or more times. This feature is supplied to Unify DataServer/ELS through the operating system; you will not be able to use this notation if your operating system does not support this type of regular expression.

For example, 'fa-z]+' matches '[a-z][a-z]*' (any alphabetic string).

{} Braces indicate repetition of the previous regular expression. This feature is supplied to Unify DataServer/ELS through the operating system; you will not be able to use this notation if your operating system does not support this type of regular expression.

m	An integer in braces matches the previous regular expression m times. The integer must be less than 255. For example, 'a{3}' matches a three times, which matches only the string aaa.
m,	An integer and a comma in braces match the previous regular expression m or more times. For example, ' b {1,}' matches b one or more times, which matches the strings b, bb, and bbb.
m,n	An integer, comma, and another integer in braces match the previous regular expression at least m but not more than n times. Both integers must be less than 255. For example, 'c{1,2}' matches c at least one but not more than two times, which matches the strings c or cc.
0,	A zero and a comma in braces have the same effect as using the asterisk. For example, 'd{0,}' matches d zero or more times. And 'hi{1,2}' matches either hi or hii.
1,	A one and a comma in braces have the same effect as using the plus symbol. For example, 'e{1,}' matches e one or more times.
( )	Parentheses are used for grouping. Grouping allows repetition operators to work on each character or an entire subexpression. For example, the regular expression '(ab){1,2}' matches the strings ab and abab. And '(hi){1,2}' matches either hi or hihi.

Enclose expressions using regular expression notation in single quotes as with STRING field legal values; for example, '[A-Z)' or 'Mac.*'. Additional examples using Regular Expression Notation include the following:

'[0-9]. [^0-9] {2}' matches a digit, followed by any single character, then two non-digit characters.

' [0-9] {3}-[0-9] {4} *' matches three digits, followed by a hyphen, four digits, then 0 or more blanks. This notation can be used to test for legal telephone numbers.

And '[A-Z] [0-9] [A-Z]...[A-C]' can be used to test for legal values of an inventory number of the general form A1B9999C.

Set all non-STRING legal values by specifying a constant, or a value "greater than" (>), "greater than or equal to" (>=), "less than" (<), "less than or equal to" (<=). Or specify the legal values as a range of constants.

FYI message statement

Use the *FYI message statement* to specify the message you want displayed when the cursor is on this field of an ENTER screen. This statement must begin with the FYI keyword. The message displays on the bottom line of the screen and can be up to 74 characters long.

Error message statement

Use the *error message statement* to specify text that displays if the value entered for a field does not match a legal value. This statement must begin with the ERROR keyword. The message displays on the bottom line of the screen and can be up to 74 characters long.

Help statement

Use the *help statement* to identify the file that displays when the user asks for help documentation on this field. You can display this file on an ENTER screen form by pressing CTRL U or TAB while the cursor is at this field.

This statement must begin with the HELP keyword followed by the name of the ASCII file. Unify DataServer/ELS looks for this file in the directory../hdoc (relative to the

DBPATH environment variable). Text displays on the screen exactly as it is entered in the help file.

You can control where the help file displays on the screen by starting the file with the statement:

@x,y

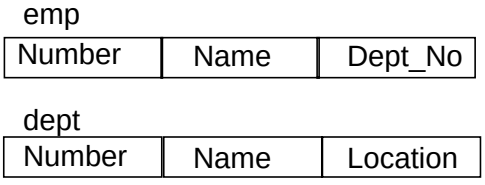
where x and y are the column and row coordinates of the file's upper left corner. This statement must be on a line by itself. ENTER screen coordinates are zero-based. The valid x (column) coordinates are 0-131, and the valid y (row) coordinates are 2-21.

If you do not include display coordinates, ENTER tries to display the file at the field's input position. If the help file is too wide to fit on the screen at the input position, ENTER displays the file in the middle of the screen. If the help file is still too wide, ENTER truncates all text that overruns the right margin. In addition, ENTER does not overwrite the last three lines on the screen. Therefore, it truncates any text that overlaps these lines.

Plan your help file and its display coordinates carefully to achieve the best results.

An example file

The following is an example of the kind of file you can develop to control default and legal values, and display messages for fields. The example database has the tables and fields shown in Figure 8.19.



8.19 Example file tables and fields

Figure 8.20 shows the database design for the example database:

Table/field	Ref	Type	LEN	Long name
emp	7			employee
*enum		NUMERIC	7	Number
ename		STRING	30	Name
depno		NUMERIC	9	Dept_No
dept	3			department
*dnum		NUMERIC	9	Number
dloc		STRING	30	Location

Figure 8.20 Example file database design

The database also has the Advanced field attributes file shown in figure 8.21.

```
record-type emp
   Number:  (unique,+200)
   Name:
      FYI='Please enter the employer's full name.'
   Dept_No: (20), \
         checkreference dept
      HELP=dept .help
      ERROR='This is an invalid department number;  \ reenter
number.'

record-type dept
   Number: (20), 20, 30, 40, 55, 60, 69, 72, 80, 85, 88, \
            90, 103, 105, 110, 120
      HELP=dept.help
      ERROR='This is an invalid department number;  \ reenter
number. '
   Location: ($BLOC), 'Chicago', 'New York', 'Bismark', \
                      'Portland', 'San Francisco', 'Miami', \
                      'Dallas', 'Denver', 'Salt Lake City', \
                      'Phoenix', 'Lexington'
      HELP=location .help
      ERROR= 'This is not a legal location, display \'help\' \
             if you need a listing. '
```

Figure 8.21 Example file advanced field attributes file

Note that the checkreference keyword is entered in the Dept_No field statement of the emp table.

Consequently, the only values that can be entered for Dept_No are existing values of NUMBER.

In this example, a table is entered for the Dept_No checkreference definition because the field, depno, does not have an explicit relationship defined with the key field, dnum.

Chapter 9: Database maintenance utilities

This chapter describes the Unify DataServer/ELS options available to ensure that your database can be recovered if it is damaged or lost. These options let you write database backups, read database backups, rebuild the hash table, rebuild explicit relationships, and define database volumes.

It is recommended that you back up your database daily, especially if it is updated frequently.Then, if a hardware failure destroys file system integrity, or a bad block develops on the disk, you can restore the database file to a recent state. Write Database Backup lets you do this.

Unify DataServer/ELS also has an option to log all transactions made between the database backups. The Read database backup option reads in the latest backup and replays the transaction log. By setting the transaction logging parameters with Transaction logging status, a more accurate recovery of your database is ensured following a system crash. For more information about Transaction logging, see section 12.1.

If a failure occurs and you suspect that database integrity has been disturbed, Unify DataServer/ELS lets you rebuild the database hash table and explicit relationships.

For example, if you suddenly cannot access a record or group of records by key, you can Rebuild the hash table (section 9.3).

Similarly, if queries or programs that use explicit relationships begin to yield inconsistent results, you can use Rebuild Explicit Relationships (section 9.4) to restore the relationships. If these methods fail, you can dump the data using SQL (Dumping data to external files, section 16.3), run Create database (section 8.3), and then load the data back in using database load (section 12.6).

9.1 Write database backup (budb)

This program writes a backup tape, or checkpoint, of the Unify DataServer/ELS database file, file.dbv, the B-tree index file, and the data dictionary.

First, the program locates the database file. Then it determines whether the file is an ordinary operating system file or spreads across several raw devices.

Write database Backup reads the size of the database from the file to calculate the number of blocks to write. It can perform multiple volume backups on large database files. Write database Backup determines the name of the backup device (a diskette drive, a cartridge, or a 9-track tape drive) from the BUDEV environment variable.

Write database Backup uses three additional environment variables: BUBLK, BUTAPESZ, and BUDRVR. Set the BUBLK environment variable to specify the size of a block on the backup device.

Set the BUTAPESZ environment variable to specify the number of blocks that fit on one tape or diskette of the backup device. If you do not set BUTAPESZ or set it to zero. Write database Backup writes data until it runs out of storage space on the current diskette or tape.

Set the BUDRVR environment variable to specify a custom version of the backup device filter to be used instead of the default device filter, BUDRVR.

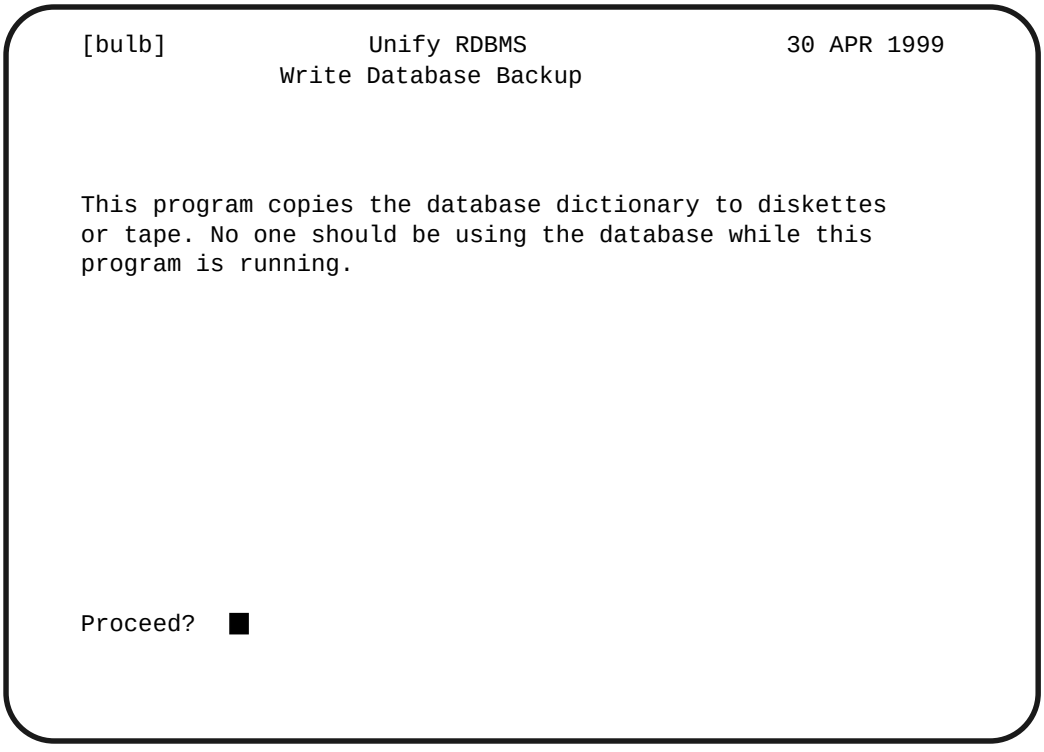
For more information on setting BUDEV, BUBLK, BUTAPESZ, and BUDRVR, see chapter 5, Environment Variables.

If transaction logging is being used, Write database Backup clears the log file when the backup is complete and resets transaction logging status to ON.

NOTE: If you have changed transaction logging from OFF to ON, you must run Write database Backup to activate a new log.

Before starting the backup, make sure no one is using the database. This ensures that a physically valid copy of the database can be made.

To start the program, select "Write database Backup" (budb) from the database Maintenance menu or the database Design Utilities menu. You see the following screen form:



Mount a diskette or tape that is blank or can be overwritten. Make sure it is write-enabled. Respond with Y or y at the PROCEED? prompt to start the backup process. If you want to exit the program and return to the menu, respond with CRTL U, N, or n.

9.2 Read database backup (redb)

This option reads the backup diskettes or tapes written by Write database Backup and then lets you replay the transaction log file produced by Transaction Logging. This procedure restores the Unify DataServer/ELS database and data dictionary file, including file. dbv and B-tree index files.

NOTE: You cannot use Read database backup to read backup files created by Unify DataServer/ELS prior to Release 4.0.

Read database Backup gets the backup device name from the environment variable, BUDEV, and the current log file or device name from Transaction Logging Status.

As does Write database Backup, Read database Backup can use the information in the BUBLK, BUTAPESZ, and BUDRVR environment variables. The BUBLK environment variable specifies the size of a block on the backup device.

BUTAPESZ specifies the number of blocks that fit on one tape or diskette of the backup device. If BUTAPESZ is not set or is set to zero, Read database Backup reads data until the end of the current diskette or tape volume.

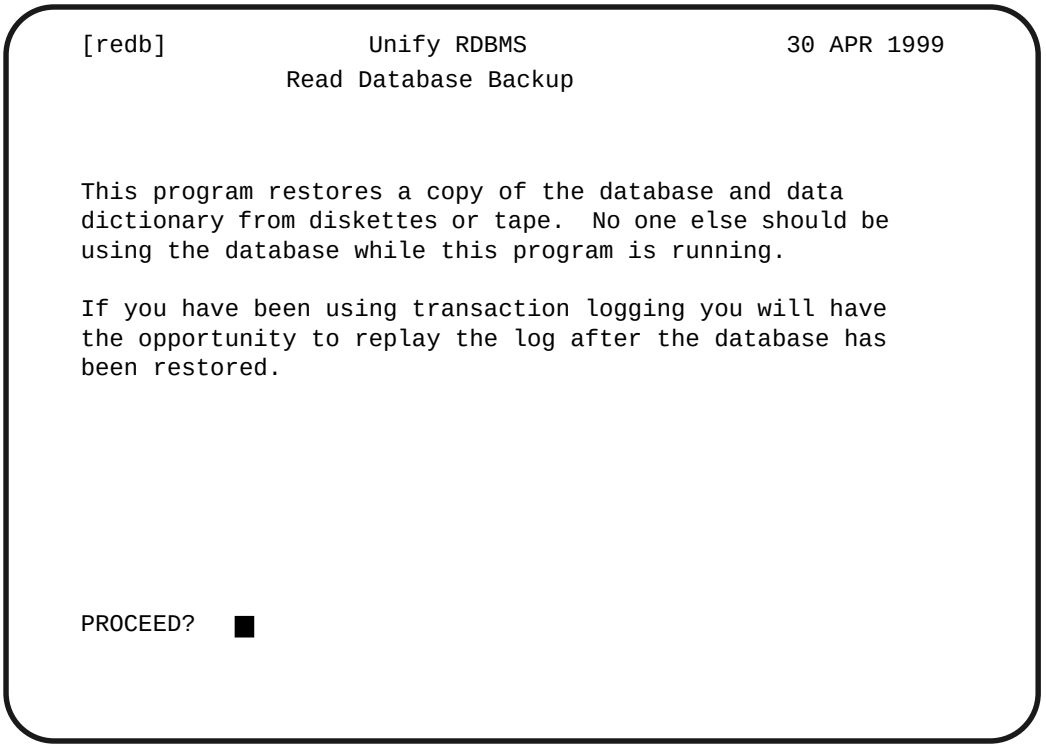
BUDRVR specifies the custom backup device filter that should be used instead of the BUDRVR program.

For more information on setting BUDEV, BUBLK, BUTAPESZ, and BUDRVR, see chapter 5, Environment variables.

NOTE: Since Read database Backup restores the data dictionary from the backup device, after the restoration is complete, you must exit Unify DataServer/ELS and login again. This step causes Unify DataServer/ELS to open the "restored" data dictionary.

Reading the database backup

Make sure no one is using the database before you start the recovery process. To start the option, select "Read database Backup" (redb) from the database Maintenance Menu. The following screen form displays:



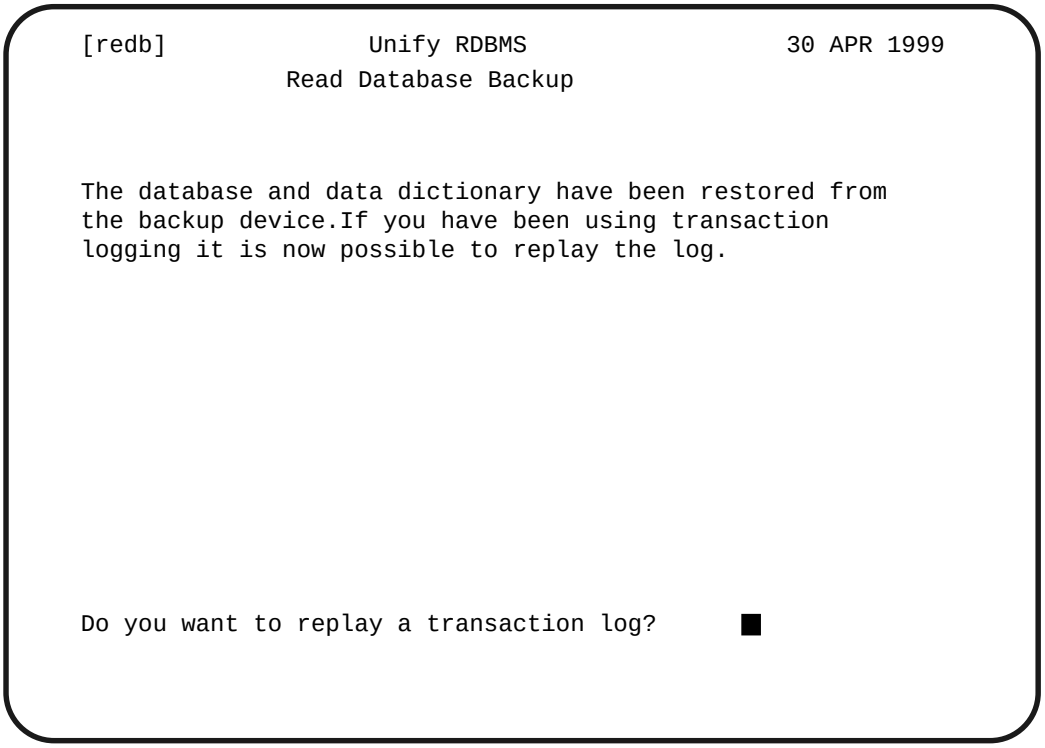
If you are using a device as the transaction logging file and want to use Read database Backup, proceed as follows:

1. Unmount the transaction log.
2. Mount the first backup diskette or tape.
3. Make sure the backup device is on line and ready.
4. Then respond with a Y to start the recovery process.

To exit the program and return to the menu, respond with CRTL U, N, or n. You can exit the program at various points after it has started. However, if you have read in one or more volumes when you exit, the database will be invalid. You must then completely recover another backup before using the database.

Replaying the transaction log

After the backup file has been successfully read, you can replay the current transaction log. You see the following screen form and prompt:



If you answer n (no), transaction logging is turned OFF or ON—depending on the current request set in Transaction logging status—and the program ends.

If you want to replay the current transaction log file to bring the database up to date, answer y (yes). The name of the log file used in the replay process is set with Transaction logging status.

The replay process reruns the transactions from the transaction journal. This is also known as *roll forward recovery* because the resulting database is built from a previous backup plus the transaction log.

This program uses the Unify DataServer/ELS transaction configuration file, unify.conf, to determine the transaction configuration. The Unify DataServer/ELS transaction configuration file is created by Transaction logging status and Read database backup when a database is recovered.

While the replay process runs, the following screen form displays:

```
[redb]              Unify RDBMS                  30 APR 1999
                Read Database Backup


PERFORMING THE TRANSACTION LOG REPLAY

BUILDING THE TABLE OF INCOMPLETE TRANSACTION









     0 TRANSACTION PACKETS PROCESSED
```

A *packet* is a database record that has been modified, deleted, or added. A *transaction* is a group of packets preceded by a "start logical transaction" code and followed by an "end logical transaction" code. The number of replayed transaction packets increment on the screen form after every hundred transaction packets have been read.

After the first pass through the transaction log is complete, the following screen form displays:

```
[redb]                    Unify RDBMS                    30 APR 1999
                     Read Database Backup


PERFORMING THE TRANSACTION LOG REPLAY

REPLAYING THE ACTUAL TRANSACTIONS

INCOMPLETE TRANSACTION GROUPS: 1
FORCED ENDLTG:  0

REPLAY ERRORS:
ADDREC:  0          DELETE: 0          PFIELD:  0

THE LAST VALID TRANSACTION GROUP ENDED ON: Mon Mar 10 9:06:23 1999

ALL nnn TRANSACTION PACKETS HAVE BEEN PROCESSED THERE WERE NOT DATABASE
ERRORS ENCOUNTERED.
```

The prompts on this screen form are described as follows:

Incomplete transaction groups:

The number of transactions found to be incomplete. These transactions have a "start logical transaction" code but do not have an "end logical transaction" code. These transactions cannot be replayed. Information about any incomplete logical transaction group is placed in the incomplete summary file. The default name for this file is replay, not.

Forced endltg:

The number of transactions forced to end by the start of another logical transaction group. This is because the program can only have one logical transaction active at a time.

Replay errors:

On the line below this prompt are error counters. The counters increment when "add record," "delete record," and "change field" errors occur. An error that occurs during the replay process is placed in the replay error file. The default error file name is replay, err.

After the transaction log has been replayed, you can back up the rebuilt database, clear the transaction log file, and restart transaction logging. The following screen form displays:

```
[redb]              Unify RDBMS                  30 APR 1999
                Read Database Backup



The transaction log has been successfully replayed and the database
is in a consistent state. It is a good idea to make a backup: of the
database at this point.

If transaction logging is ON you will now have the option to backup
the database and restart transaction logging. If you do not choose to
do this, transaction logging will be turned OFF until the next time
you backup the database.




Do you want to back up the database and restart
transaction logging?
```

A n (no) answer turns transaction logging OFF and ends the program.

A y (yes) answer calls Write database backup, so you can write a new backup. After the backup is complete, transaction logging is turned ON.

NOTE: After Read database backup is complete, be sure to exit Unify DataServer/ELS and login again. This step causes Unify DataServer/ELS to open the "restored" data dictionary.

9.3 Rebuild the hash table (rekey)

This option completely rebuilds the hash table. Use this option when you cannot access a record by its primary key, but you can find the record by scanning the database table or by using a B-tree index. This can happen if you change the internal data type of a key field—using Modify database Design—and then reconfigure the database.

Examples of modifying the internal data type are changing a NUMERIC 3 field to NUMERIC 9 (from short to long), or changing an AMOUNT 2 to AMOUNT 12 (from long to double).

This maintenance option zeros the hash table index, then reads the database, making an entry for each key it finds. The program displays neither selection options nor other output.

Start the program by selecting "Rebuild the Hash Table" (rekey) from the database Maintenance Menu. You see the following screen form:

```
[rekey]              Unify RDBMS                  30 APR 1999
                Rebuild the Hash Table



This program rebuilds the database hash table index.  No one
else should be using the database while this program is
running









PROCEED? ■
```

Respond to the prompt with Y or y to start the program, or CRTL U, N, or n to exit to the menu. When the program is finished, it displays the following screen and message:

```
[rekey]                    Unify RDBMS                30 APR 1999
                      Rebuild the Hash Table




 Total records:nnn
 Completed records:nnn

 Total tables:nnn
 Completed tables:nnn

 Current table:xxx
 Number of records:nnn




  Writing hash table
  Program complete. ->->
```

Any keyboard response returns you to the menu.

9.4 Rebuild explicit relationships (repoint)

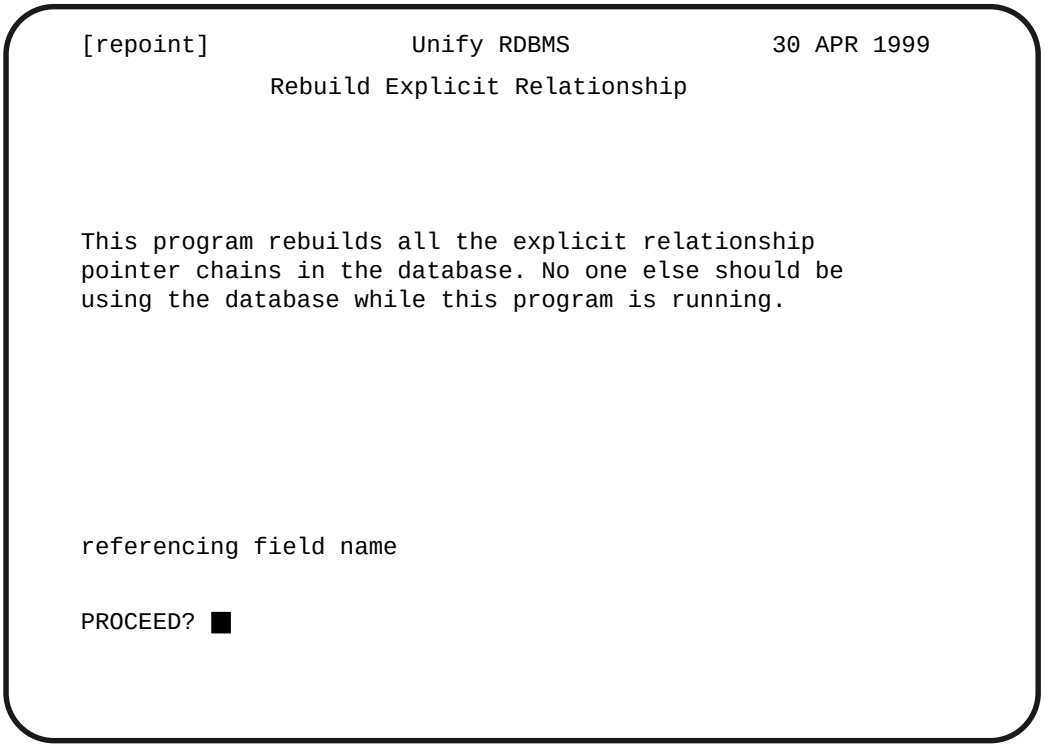
This option rebuilds the explicit relationships in the database. Use this option if you change an existing ordinary field to an explicit relationship or if an explicit relationship is damaged. To rebuild explicit relationships,run repoint from the shell.

If you have a large database, and only a few explicit relationships need to be rebuilt, see Rebuilding selected explicit relationships at the end of this chapter.

Rebuild explicit relationships zeros the existing pointers, then reads the database, relinking the related records. It can do this because Unify DataServer/ELS stores a certain amount of redundant data that indicates what the relationships should be.

From the Menu Handler

From the database Maintenance menu, start the program by selecting Rebuild explicit relationships (repoint). You see the following screen form:



Respond to the prompt with Y or y to start the program, or CRTL U N, or n to exit to the menu. When the program is finished, it displays the message:

Program complete. -> ->

Any keyboard response returns you to the menu.

If the repoint program cannot relink some of the records, the following message displays instead:

— WARNING —

All the records were not relinked. You must manually relink them, following the instructions given in the file "repoint.err".

This message means repoint cannot repair the pointers for some of the database records because the pointer values linking the records to their reference files are invalid, and the data values in the reference fields do not exist in the reference files.

Two reasons for invalid pointer values can be that the record has been corrupted by a bug in a user program, or there is a hardware malfunction. Another reason is that a valid data value has never been stored in an explicit relationship field.

For example, when a child record has only a system default (null) value stored in a field that references a parent record, the link to the parent record is never established. This can happen if you add a record, but do not enter a value in a field that does not have an Advanced field attribute default value (see section 8.6, Advanced field attributes).

If you want to use the system default (null) value in the child record, you can try the following:

1. Add a parent table record that contains the same value in its referenced field as the default value in the child record referencing field.
2. Run Rebuild explicit relationships to establish the link between the parent and child records.

If you run Rebuild explicit relationship on a database for which explicit relationship fields have never been set, the previous warning displays.

Your current directory might show a file called repoint.err, which lists the records that repoint can't fix. The repoint.err file contains the following information:

Table — The name of the table that has the bad pointer. It is the *child* table in the relationship.

Location — A number indicating the relative location of the problem record in a set of records in the same table.

The table name and location number can be used with the database Test driver setloc function, to find the record. For information about setloc, see database Test driver in the *Unify DataServer/ELS HLI Programmer's Manual*.

Related Table The name of the related table. It is the *parent* table in the relationship.

Reference Field

The name of the explicit relationship field in the *child* table. This is the field that needs to be updated to relink the record to the list for the *parent*.

This field can be updated using the database Test driver setloc function to find the record. For more information on setloc and record relinking, see database Test driver in the *Unify DataServer/ELS HLI Programmer's Manual.*

The repoint.err file may also contain the following message:

nnn child records with null reference field values and no matching parent record.

This message lets you know that some child records have default or null values for the referencing field.

From the operating system

From the operating system shell, you can rebuild explicit relationships by running the REPOINT executable. This makes it possible for you to use a command script executed by cron to run REPOINT during DBMS down-time like in the middle of the night or on the weekend.

To rebuild all explicit relationships, you can run REPOINT from the shell using the following command:

```
REPOINT -a
```

where -a tells REPOINT to rebuild the pointer lists for all the explicit relationships in the database. (See the next subsection if you want to rebuild only selected explicit relationships.)

Do not omit the -a argument. If you omit the -a, REPOINT assumes it is being run from the Menu Handler, and displays the screen shown in section 9.4.

As REPOINT rebuilds each explicit relationship's pointer list, it sends the name of the current relationship's referencing field to the standard output device. If REPOINT cannot relink all of the records, it displays the following message after the field names:

— WARNING —

All the records were not relinked. You must manually relink them, following the instructions given in the file "repoint.err".

When you run REPOINT from the shell, it returns an exit status to indicate whether it completed successfully. REPOINT returns a 0 if it completes successfully. If REPOINT returns a non-zero exit status, check the error log for messages.

NOTE: REPOINT keeps the database locked while running. If you run REPOINT from the shell, no other process can update the database until REPOINT finishes.

Rebuilding selected explicit relationships

If you change an existing ordinary field to an explicit relationship, or if an explicit relationship has been damaged, you must rebuild the corresponding pointer lists.

You do not always need to rebuild all the explicit relationships in the database. And for very large applications, you may not be able to rebuild all relationships because of the time required. For these reasons, you can have REPOINT selectively rebuild only specified explicit relationships.

To rebuild selected explicit relationships, run REPOINT from the operating system shell, giving the referencing field names as command line arguments. The following simple example illustrates the steps involved:

1. Print a database design report. This report lists the current database design, including a list of table and field names. See Print database Design in section 11.1.
2. Note the Table and the reference field names from the Schema listing section of the Data dictionary report. Figure 9.2 shows an example Data dictionary report.

```
TABLE/FIELD        REF
orders
   order_no
   order_dt
   order_cn         cust_no
```

where: orders is the child table containing the reference field; order_cn is the reference field; and cust_no is the key field in the related table (parent).

Figure 9.2 Table and fields from data dictionary report

3. Access the operating system shell by entering sh at the selection prompt. (See Running Operating System Commands from Unify DataServer/ELS in chapter 2.)
4. Type the following command at the shell prompt, using the unique *short* field names only (you cannot use long field names):

   REPOINT order_cn

   As REPOINT runs, it prints the name of the current relationship's referencing field. If REPOINT cannot relink all of the records, it displays the following message after the field names:

   — WARNING —

   All the records were not relinked. You must manually relink them, following the instructions given in the file "repoint.err".

   You can give REPOINT a list of reference field names. REPOINT rebuilds the relationship pointers for each field in the list and prints the field's name as it is being processed. To run REPOINT for a list of reference field names, use the following command line format:

   REPOINT ref_field1 [ref_field2 ref_field3 ...]

   where ref_field1, ref_field2, and ref_field3 are the name of referencing fields.

9.5 Define database volumes (volmnt)

Unify DataServer/ELS lets you store a database in up to eight database volumes. A volume is a single area of storage space for database information.

The *root volume* is the first volume, or volume zero, because it stores the database header. This header includes database parameters and the hash table. The root volume must always have an offset of zero blocks from the starting address of the root volume.

Other volumes store database information. These volumes can have volume numbers from one through seven. You use Define database volumes to describe database volumes to Unify DataServer/ELS.

In Unify DataServer/ELS, database volumes can be either ordinary operating system files (file volumes) or *raw files* (device volumes). Raw files consist of contiguous blocks of disk space, usually called *disk partitions*.

Normally, disk partitions are formatted into file systems using the mkfs command. These file systems are then mounted as directories in the operating system file hierarchy.

However, instead of formatting a partition and using it as a file system, you can leave the partition unformatted and use it as storage for a Unify DataServer/ELS database. Unformatted disk space is called a *raw partition*.

A raw partition can be accessed directly by Unify DataServer/ELS's access routines without using the operating system file access software. This increases the storage efficiency of a Unify DataServer/ELS database.

The diagram in Figure 9.4 illustrates how a system with two disks can be configured using Unify DataServer/ELS with device volumes..

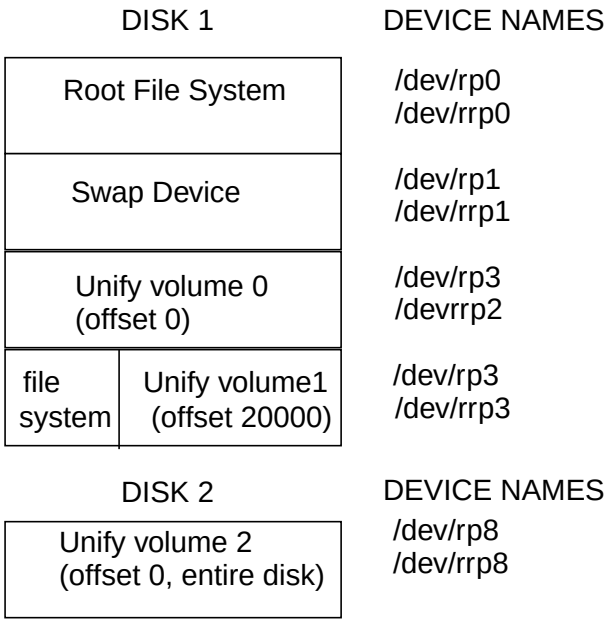


Figure 9.4 Device volumes on two disks

On the first disk, device rp0 contains the root file system. Device rp1 contains the swap area. Device rp2 holds the root volume (volume 0) of the Unify DataServer/ELS database. Unify DataServer/ELS uses both the block device (/dev/rp2) and the character device entries (/dev/rrp2).

The first part of device rp3 contains a file system. It was created by running mkfs with the appropriate size parameter. The second part of the device holds volume I of the Unify DataServer/ELS database.

The entire second disk (/dev/rp8) is Unify DataServer/ELS volume 2 of the Unify DataServer/ELS database. Unify DataServer/ELS volume 2 contains the remainder of the Unify DataServer/ELS database.There can be any number of Unify DataServer/ELS databases on a given computer system. Some databases can be on raw devices and others in ordinary operating system files. Use Define database Volumes to assign the devices to volumes of a Unify DataServer/ELS database. You should assign devices so collisions are prevented.

If you specify no devices, Create database creates the file for the root volume as an ordinary file. The default name for this file is file.db. If you want to name this file something else, you can specify a different name for the file with the Unify DataServer/ELS environment variable DBNAME (chapter 5).

If you specify device volumes, Create database and reconfigure database create file.db and file.dbr as special files. In this case, file.db is the block device and file.dbr is the character device for the root database volume.

Using a raw disk partition

To convert an existing database stored in an ordinary file to one stored on a raw partition, take the following steps:

1. Log in as the operating system super user.
2. Select a file system large enough to hold the current file.db. When you convert to a raw partition, the entire database (file.db) must initially fit on one volume. Later, the database can expand to additional volumes.

   If the selected file system is mounted, you must unmount it. If the *mount* command is in the /etc/rc command script, you must also remove the mount command for this partition.
3. Make sure that the /dev directory contains the special files for the partitions you want to use. If the device files don't exist, create them using mknod. Make sure that the owner is root and that its mode is 666.
4. Log in to your database as the Unify DataServer/ELS super user.
5. Use Define database volumes (volmnt) to define the database volumes. Include volume names, offsets, and lengths, and device names.
6. Back up the database using Write database backup (section 9.1). This backs up the new volume information.
7. Run Create database (section 8.3). This sets up file.db and file.dbr as special files pointing to the device defined by $DBPATH for the root database volume.
8. Escape to the shell and use chmod, chown, and chgrp to set correct read and write file permissions for the volume devices specified in Step 5. Also set the appropriate permissions for file.db and file.dbr.

9. Restore the database. Use Read database backup (section 9.2) to read in the backup created in Step 6.
10. While still logged in as the super user, run Reconfigure database (section 8.4). This stores the new volume information in the file.db header so additional volumes can be used as the database expands.

To display information about a raw database volume, run the database Statistics report (dbstats).

Using ordinary operating system files

Unify DataServer/ELS lets you use ordinary operating system files for database storage. There are two ways to do this: with no database volumes defined or with database file volumes.

When you use ordinary files without database volumes defined, the database is created as an ordinary ASCII file. The default name for this file is file.db. However, if you want to call the file something else, you can specify a different name in the Unify DataServer/ELS environment variable DBNAME.

The default storage location for the database file is the current database directory. You can change the location for the database file by setting the DBPATH environment variable. For more information about setting Unify DataServer/ELS environment variables, see chapter 5.

File volumes

When you use ordinary files with database volumes defined, the volumes are ordinary files, or file volumes. With *file volumes*, you can have several database volumes in one or more file systems.

To locate file volumes, Unify DataServer/ELS checks the directory specified in DBPATH. If DBPATH is not set, Unify DataServer/ELS checks the current directory. If Unify DataServer/ELS cannot locate the file volumes in either directory, it checks the directories listed in the VOLPATH environment variable.

A file volume must have an identifying suffix which tells Unify DataServer/ELS which database volume it is. This suffix is .db#, where # is the volume number. The volume

number is the same as the volume's line number on the Define database volumes screen form. For example, the root volume has line and volume number zero, and the next volume has line and volume number one.

If you do not include the suffix when defining a volume device, Define database volumes adds the suffix to the file name, using the volume line number in the .db# suffix.

Except for the root volume, files to be used as file volumes must exist before you define them on the Define database volumes screen. Define database volumes verifies that these files exist. For non-root file volumes, the program does not let you enter the name of a file that it cannot find in the directory specified in DBPATH, the current directory (if DBPATH is not set), or the directories specified in VOLPATH.

The example in Figure 9.5 illustrates how a file system can be used to store a Unify DataServer/ELS database with four file volumes..

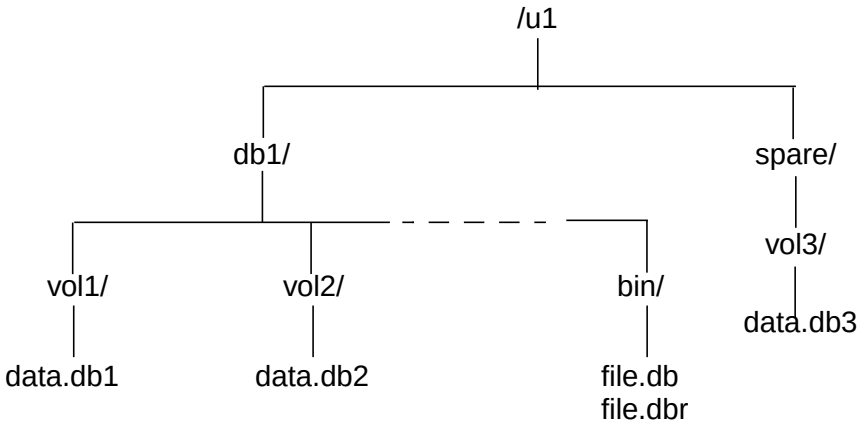


Figure 9.5 File system structure

In this example, the DBPATH and VOLPATH environment variables are set as follows:

```
BPATH = /ul/dbl/bin
VOLPATH = /ul/dbl/voll:/ul/dbl/vol2:/ul/spare/vol3
```

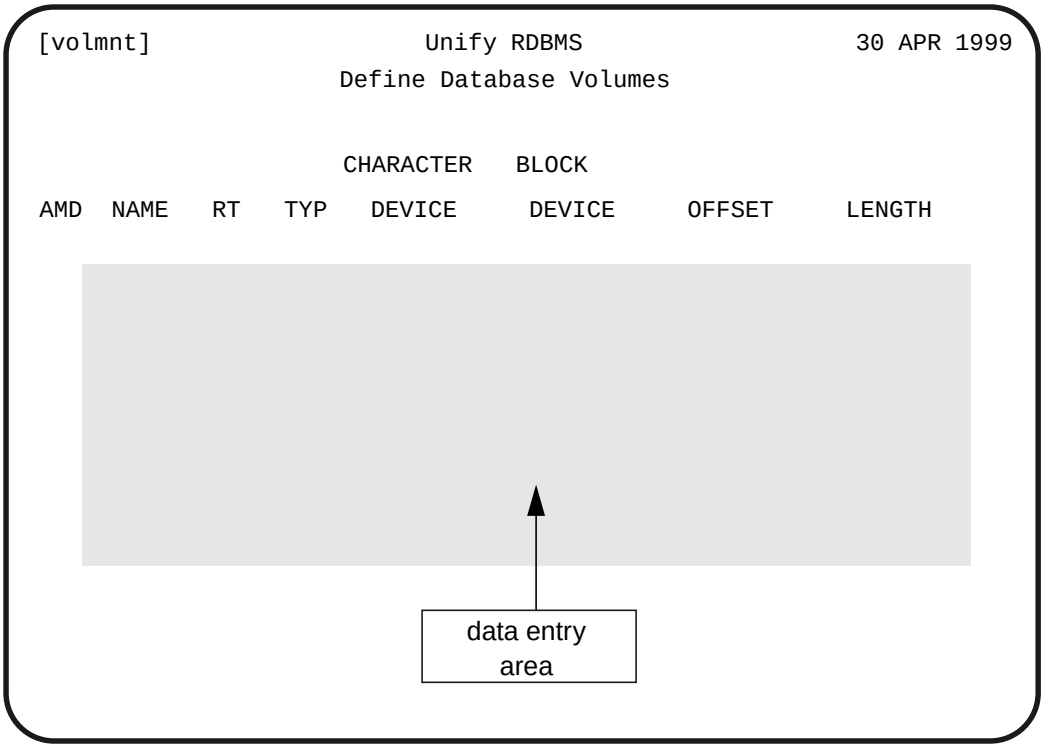
Figure 9.6 summarizes the difference between file volumes and device volumes.

File volumes...	Device volumes...
are stored in a file system as ordinary files	are stored on raw disk space
can only be accessed using the operating system file access runtiness	can be accessed directly by the Unify access routines without using the operating stem access routines

Figure 9.6 Comparison of file versus device volumes

Defining database volumes

Start the program by selecting "Define database volumes" (volmnt) from the database Maintenance menu. The following screen form displays:



Each line in the data entry area represents one volume of the database. The prompts on this screen form are as follows:

AMD — The current mode: A(dd), M(odify), or D(elete). To add a volume, move the cursor to an empty line and enter an a in this column.

To modify a volume, move the cursor to the line for that volume and enter an m.

To delete a volume, move the cursor to the line for that volume and enter a d.

To move to the next line, press RETURN. To move to the previous line, press CRTL U. Press RETURN on the last line to move to the first line. Press CRTL U on the first line to exit the program.

NOTE: Once you have defined a volume, you cannot exit the program unless one volume is defined as the root or all volumes are deleted.

NAME — A unique four-character name for the volume. This name is used to order the volumes alphabetically on the screen and in the database header.

The order volumes appear on the screen is the order they are used to store data. The volume name is also used to report errors.

RT — An x in this column marks this as the root volume. The root volume contains the database header, which includes the hash table and database parameters. You must define a root volume, and it must have an offset of zero blocks.

Only one volume can be a root device. Therefore, you cannot enter an x in this column if another entry is marked as the root.

The root volume displays on the first line (line 0) in the data entry area, regardless of its volume name. All other volumes are sorted by volume name. The root volume is volume 0, the next is volume 1, and so forth.

TYP — The volume type. Enter F or f for a file volume. Enter D or d for a device volume. You can have a mix of file and device volumes.

For device volumes:

If you enter D to TYP, you must enter the name of an existing character or block device in the CHARACTER DEVICE column. You must also enter the name of an existing block device in the BLOCK DEVICE column. These devices should be in the /dev directory.

For file volumes:

If you enter F to TYP to select a file volume, the cursor skips to the BLOCK DEVICE column. Enter the name of an existing regular file, with or without the .db# suffix.

Because a file volume uses the same file for both the block and character devices, the program automatically places the block device name in the CHARACTER DEVICE column.

If this volume is the root, the program fills in the BLOCK DEVICE and CHARACTER DEVICE names. Both names are either the default volume name (file.db) or the name specified in DBNAME, if it is set.

Character device

The character device name for the volume.

For device volumes:

The device name can be up to 18 characters. Of the 18 characters, five are for the device prefix, /dev/, and 13 are for the device subdirectory path. If the device name you enter does not begin with /dev/, the prefix /dev/ is inserted at the beginning of the device name.

The character device must exist in the /dev directory. It must be either a character or a block special file. If the file name you enter doesn't exist or is of the wrong type, you are requested to enter another device name.

If you want to use a file name that does not exist in /dev, you must use the super user ID and create the file with the mknod command before you define the database volume that uses it.

For file volumes:

This field is automatically filled in by the Define Database Volumes program. This name is the same as the name you entered in the BLOCK DEVICE column.

Block device

The block device name for the volume.

For device volumes:

The device name can be up to 18 characters. Of the 18 characters, five are for the device prefix, /dev/, and 13 are for the device subdirectory path. If the block device name you enter does not begin with/dev/, the prefix /dev/is inserted at the beginning of the device name.

The block device must exist in the /dev directory. It must be a block special file. If the file name you enter doesn't exist or is of the wrong type, you are asked to enter another device name.

If you want to us a block device that does not exist in /dev, you must use the super user ID and create the file with the mknod command. You must create this file before you define the database volume that uses it.

For file volumes:

The block device name can be up to 17 characters. Of the 17 characters, four are for the file suffix, .db# where # us an integer volume number from 1 through 7. The remaining 13 characters are for the file name.

The name you enter in the BLOCK DEVICE column must be the name of an ordinary file. If the volume is the root volume, the program automatically fills in the volume name. The default root volume name is file.db or the names specified in DBNAME, if that environment variable is set.

If the volume is not the root, the ordinary file must exist in the directory specified in DBPATH, in the current directory (if DBPATH is not set), or in the directories specified in VOLPATH.

If the file name you enter does not end with .db#, the suffix .db# is added to the file name. The # in the suffix must match the volume number, which is the same as the volume's line number on the Define database Volumes screen.

For example, the first volume after the root volume is volume 1 and has the suffix .dbl. Volume 2 has the suffix .db2, and so on. The exception is the root volume. The root is volume zero, so it uses the .db suffix.

Once you enter a valid block device name, the program automatically places that name in the CHARACTER DEVICE column.

Offset

The offset (the number of 512-byte blocks) from the start of the device where Unify DataServer/ELS data storage begins. You specify offsets only for device volumes. File volumes always have an OFFSET of 0.

For example, the device can be a disk partition that is partitioned for more than one database volume. If the first volume begins at location 0 and ends at 999 (999 512-byte blocks), its OFFSET would be 0. OFFSET for the second volume would be 1000.

NOTE: If you enter an x in RT, OFFSET defaults to 0 and you cannot change it.

Length

The length (the number of 512-byte blocks) of the volume.

If volume type is F, the length you enter refers to the maximum size the file volume will grow before it is considered full. If the volume type is D, the length you enter refers to the current size of the device volume's disk area.

Length must include an allowance for the file system overhead needed to maintain the file volume, such as indirect blocks.

After making changes to any of these definitions, you must reconfigure the database in order for the changes to take effect.

Part III: System administration

Part III describes the options on the "Security Maintenance" and "Data Dictionary Reports" menus, plus several miscellaneous programs. You can use these programs to control database security, provide database reports and database design information, report the status of database changes, develop help documentation and command scripts, register executables, and load data in the database.

Chapter 10: Database security utilities

This chapter describes how to control access to a Unify DataServer/ELS database to create a secure environment. Using an encrypted password scheme, Unify DataServer/ELS has several levels of protection. These Unify DataServer/ELS protection levels enforce login security, program and menu security, and field security.

Login level security prevents unauthorized operating system users from accessing Unify DataServer/ELS databases. Users must enter user ID's and passwords to access databases. The system administrator can be assigned a super user ID.

Program and menu level security restricts unauthorized database users from accessing some programs or menus. If users don't have access privileges for specific programs or menus, those options don't display.

Field level security prevents unauthorized access to individual database fields. Users must supply the correct read and write passwords to access read-protected or write-protected fields.

You can combine Unify DataServer/ELS's protection with your operating system's file protection to increase security. The operating system file protection scheme can prevent users from accessing the database contents through the shell.

The following chapters explain how to use these features. The last of these chapters describes how to use the Unify DataServer/ELS security system with file protection.

10.1 Modify system parameters & security (parmnt)

This program maintains basic parameters for the menu system. You can change information about the super user, the system title, and mnemonic codes for the months of the year. You can also set the system entry point and login screen display. Because this option can be used to change the super-user ID and password and the system login screen, you should use Add or modify individual privileges to restrict who may access it (section 10.3).

To run this program, select "Modify system parameters & security" (parmnt) from the Security maintenance menu. The screen form displays as follows:

```
 [parmnt]              Unify RDBMS                30 APR 1999
            Modify System Parameters & Security

SUPER USER ID     : su

PASSWORD          :

ENTRY POINT       : mainmenu

SYSTEM HEADING    : UNIFY RDBMS

MONTH MNEMONICS   : JANFEBMARAPRMAYJENJULAGUSEPOCTNOVDEC

LOGINFLAG (T/F)
```

The entries on the screen are system parameters you can change. The system parameter prompts are described as follows:

SUPER USER ID

The login ID of the super user. It defaults to su. To change the password, enter any string of up to four characters.

PASSWORD The super user password. It defaults to su. To change the password, enter any string of up to eight characters.

The string does not display as you enter it, so the program asks you to re-enter the password to confirm the spelling. The program encrypts the password before storing it in the data dictionary.

ENTRY POINT

The name of the menu the super user sees after logging in. Users who don't have group or individual entry points assigned also see this menu after login.

SYSTEM HEADING

A string of up to 50 characters that displays at the center of the first line on each screen form and menu in the application system.

MONTH MNEMONICS

A 36-character string that contains three characters for each month of the year. The string is used in the date that displays in the upper right corner of each screen form and menu in the application system.

LOGINFLAG (T/F)

A flag that controls whether a login screen displays when a user starts Unify DataServer/ELS. The default mode is F (false) for no login screen.

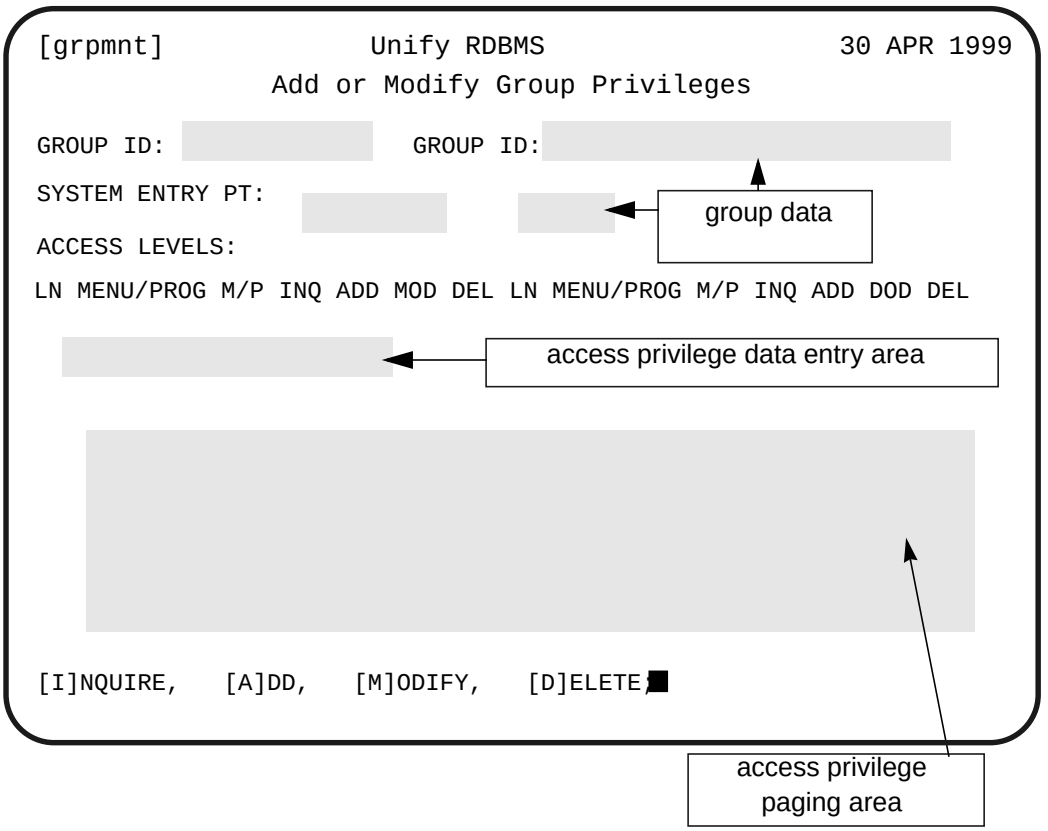
When the screen displays for this program, the cursor is at the SUPER USER ID prompt. Press RETURN to move the cursor down the screen form or CTRL U to move it up. To return to the menu, press CTRL U at the first prompt.

10.2 Add or modify group privileges (grpmnt)

This program enables you to add, modify, and delete group access privileges in the data dictionary. These privileges specify which programs and menus a group may use, as well as the types of updates they may make.

You can also set access privileges for SQL and for the Unify DataServer/ELS *exit to the operating system* command, sh. Users may access only the menus, programs, and screen forms listed for their group.

To run this program, select "Add or Modify Group Privileges" (grpmnt) from the Security Maintenance menu. The following screen form displays:



This screen form lets you inquire about, add, modify, and delete user groups and their access privileges. It has a group data entry area where you enter user groups, an

access privilege data entry area where you enter access privileges, and an access paging area where the screen form lists access privileges.

The following is an explanation of each of the prompts and column headings on the screen form:

[I]NQUIRE, [A]DD, [M]ODIFY, [D]ELETE:

The operation prompt. The meaning of each selection is as follows:

i	Inquire mode. This lets you view the access privileges for an existing user group.
a	Add mode. This lets you add a user group and its access privileges.
m	Modify mode. This lets you change the access privileges of a user group.
d	Delete mode. This lets you delete a user group and its access privileges.

GROUP ID A code of up to four characters to identify the user group.

NAME A name of up to 30 characters used for user group identification and documentation only.

SYSTEM ENTRY PT

The system name of the menu that users in this group see after they log into Unify DataServer/ELS, if they do not have a different individual entry point specified.

[N]ext page, [P]rev page, [A]dd line, or number:

The paging prompt. This prompt lets you choose the operation mode for the paging area. The meaning of each selection is as follows:

n	Displays the next page of access privileges
p	Displays the previous page of access privileges

a	Moves the cursor to the access privilege data entry area so you can add access privileges
1-999	Accesses the privilege on line nnn and positions the cursor at that line so that you can modify or delete it

column to the left of LN

Because of space restrictions, there is no name for this column. But it is the same column as the CMD column on the table entry screen. This is where you enter operation commands for the current access privilege. The cursor position marks the current access privilege.

In the paging area, you can move the cursor up or down this column by pressing CTRL U or RETURN. The valid operation commands are as follows:

m	Modify the INQ, ADD, MOD, or DEL entries for the current access privilege
d	Delete the current access privilege
q	Quit the current entry mode and redisplay the paging prompt

LN — A line number generated by the system. Use the line number to refer to the access privilege you want to modify.

MENU/PROG

The name of a menu, program, ENTER screen, or SQL screen that users in this group can access.

M/P — A display-only field that indicates the type of MENU/PROG entry. The definitions of the characters that display in this column are as follows:

M	A menu
P	A program
E	An ENTER screen

S	An SQL screen

INQ	A Y in this column indicates that users in this group can use inquire mode with the named option. An N indicates they cannot.
ADD	A Y in this column indicates that users in this group can use add mode with the named option. An N indicates they cannot.
MOD	A Y in this column indicates that users in this group can use modify mode with the named option. An N indicates they cannot.
DEL	A Y in this column indicates that users in this group can use delete mode with the named option. An N indicates they cannot.

If you have specified group or individual access privileges for users, they can access only the menus, programs, and screens listed for them on the Add or modify group privileges and Add or modify individual privileges screen forms.

For example, if you want users to have access to sh or edit, you must explicitly specify those program names on the Add or modify group privileges and Add or modify individual privileges screen forms.

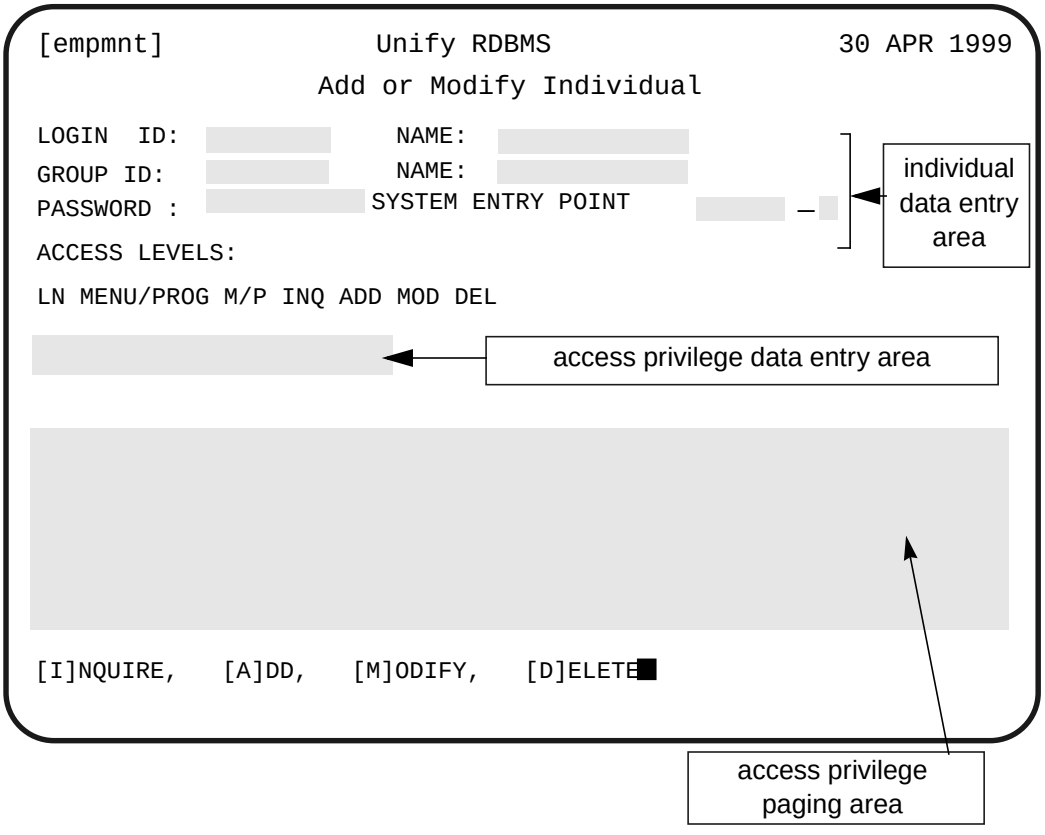
10.3 Add or modify individual privileges (empmnt)

This program enables you to maintain access privileges for individual users. You can add new privileges, or modify and delete existing privileges.

In addition, you can maintain a list of the differences in access privileges of the individual versus those of the group. Where an option or menu access item appears for an individual, it supersedes the group item.

You can set access, inquire, add, modify, and delete privileges for Unify DataServer/ELS programs, menus, screen forms, SQL, and the Unify DataServer/ELS *exit to the operating system* command, sh.

To start the program, select Add or modify individual privileges (empmnt) from the Security maintenance menu. The following screen form displays:



When you enter access privileges on this screen, you can specify differences in privileges between the user and the group. The privileges of the group determine the basic privileges for the user. You can override the group privileges if you use this program to specify personal privileges for the individual.

The screen form has an individual data entry area where you enter the individual, an access privilege data entry area where you enter the access privileges, and an access paging area where the screen form lists access privileges. The screen form prompts and column headings are described as follows:

[I]NQUIRE, [A]DD, [M]ODIFY, [D]ELETE:

The operation prompt. The meaning of each selection is as follows:

i	Inquire mode. This lets you view the access privileges for a user.
a	Add mode. This lets you add a user and the user's access privileges.
m	Modify mode. This lets you add, modify, and delete access privileges for an user.
d	Delete mode. This lets you delete a user and the user's access privileges.

LOGIN ID A code of up to four characters to identify the individual. The user enters this code to log in to Unify DataServer/ELS at the login screen.

NAME A string of up to 30 characters for the individual's name. This is for documentation and identification only.

GROUP ID A code of up to four characters to identify the individual's group.

NAME A display-only field that shows the name of the individual's group.

PASSWORD A password code of up to eight characters that the user must enter at the login screen to access the Unify DataServer/ELS system. The password does not display when you define it or when the user enters it. Unify DataServer/ELS encrypts the password before storing it in the

data dictionary or before searching for a matching user password in the data dictionary.

SYSTEM ENTRY FT

The name of the first menu the individual sees after login. It defaults to the System entry point of the group, but it can be changed for the individual.

[N]ext page, [P]rev page, [A]dd line, or number:

The paging prompt. This prompt lets you choose the operation mode for the paging area. The meaning of each selection is as follows:

n	Displays the next page of access privileges
p	Displays the previous page of access privileges
a	Moves the cursor to the access privilege data entry area so you can add access privileges
1-999	Accesses the privilege on line nnn and positions the cursor at that line so you can modify or delete it

column to the left of LN

Because of space restrictions, this column has no name. But this column is the same as the CMD column on the table entry screen. This is where you enter operation commands for the current access privilege. The current access privilege is indicated by the position of the cursor.

In the paging area, you can move the cursor up or down this column by pressing CTRL U or RETURN. The valid operation commands are as follows:

m	Modify the ACCESS, INQ, ADD, MOD, or DEL settings for the current access privilege
d	Delete the current access privilege

q — Quit the current operation mode and redisplay the paging prompt

LN — The system generates this line number. Use it to refer to the access privilege you want to modify.

MENU/PROG — The name of a menu or program that the individual can access differently from other users in the group.

M/P — This is a display-only field that indicates the type of MENU/PROG entry. The definitions for the character codes are as follows:

- M — A menu
- P — A program
- E — An ENTER screen
- S — An SSQL screen

ACCESS? — A Y in this column indicates that this individual can access the named option. An N indicates this user cannot.

INQ — A Y in this column indicates that this individual can use inquire mode with the named option. An N indicates this user cannot.

ADD — A Y in this column indicates that this individual can use add mode with the named option. An N indicates this user cannot.

MOD — A Y in this column indicates that this individual car use modify mode with the named option. An N indicates this user cannot.

DEL — A Y in this column indicates that this individual car use delete mode with the named option. An N indicates this user cannot.

If you have specified group or individual access privileges for users, they can access only the menus, programs, and screens listed for them on the Add or modify group privileges and Add or modify individual privileges screen forms.

For example, if you want users to have access to sh or edit, you must explicitly specify those program names on the Add or modify group privileges and Add or modify individual privileges screen forms.

10.4 Field security maintenance (fldsec)

Field level security is implemented in two steps:

1. Field security maintenance

   The Field security maintenance program lets you specify separate read and write passwords for each database field.

2. Process field passwords

   After you set or change a password with the Field security maintenance program, you must process the passwords, using Process field passwords, to create a binary file containing the password information. After the field password has been processed, users can only read or write to the field if they enter the correct password.

The table in Figure 10.1 summarizes how read and write passwords control access to fields.

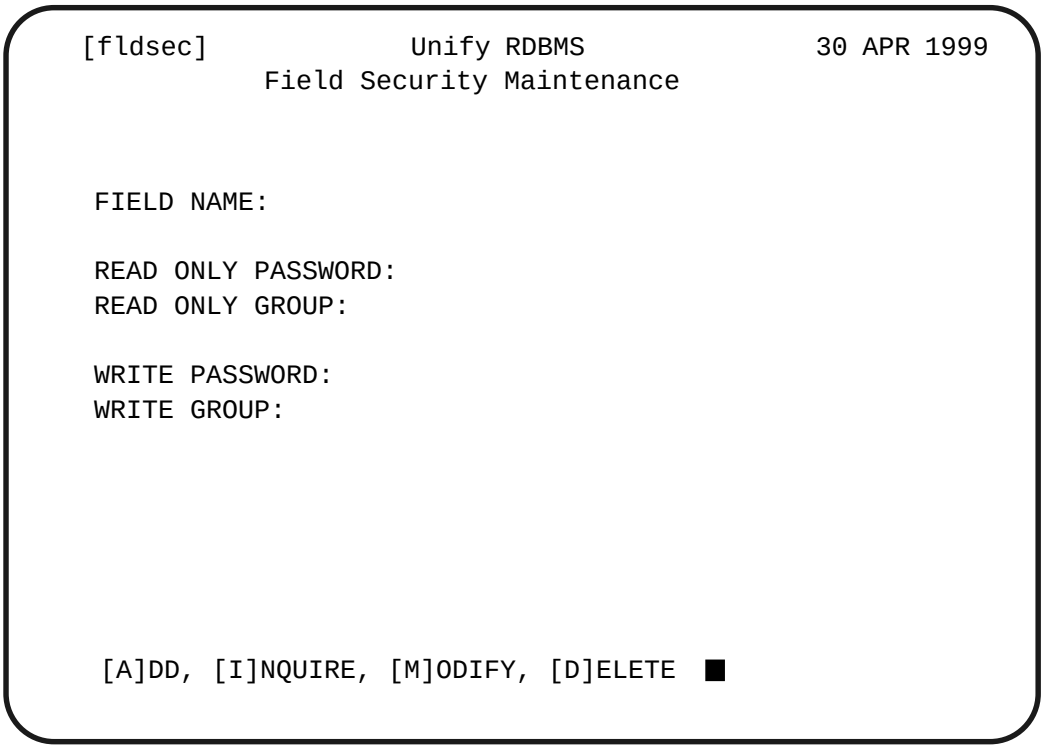
If...	Then...
you specify a read-only but not a write password	users who enter the password can both read and write.
you specify both passwords	users who enter only the write password can both read and write. users who enter the read password can only read.

Figure 10.1 Password read-write privileges

In addition to the passwords, you can assign fields a security group for both the read and write permissions:

> For a user program to add records to or delete records from a file, the program must have write privileges for all the fields in that file.
>
> However, users can access password-protected fields through ENTER and the database test driver, sys920. Neither ENTER nor the database test driver recognizes field-level security. For more information on ENTER, see chapter 15; for the database test driver, see the *UNIFY Data/Server/LES HLI Programmer's Manual.*

To start the program, select "Field security maintenance" (fidsec) from the database Maintenance menu. The following screen form displays:



The operation characteristics of this screen form are identical to an ENTER screen (see chapter 15). The prompts on the screen form are described as follows:

[A]DD, [I]NQUIRE, [M]ODIFY, [D]ELETE

The operation prompt. This prompt can vary depending on the user's access privileges.

FIELD NAME:

The name of the field for which you want to set password permissions.

READ ONLY PASSWORD:

The password that users must enter to read the field you specify in FIELD NAME. If you do not set a write password as well, this password grants users both read and write privileges for the field.

READ ONLY GROUP:

The group name for a set of fields. All fields with the same read-only group string are in the same group. If users enter the read-only password for any member of a field group, access is granted as if the passwords were entered for all fields in the group.

WRITE PASSWORD:

The password that users must enter to write to the field you specify in FIELD NAME. If you set the write password, then users who enter the password can both read and write to the field.

WRITE GROUP:

The group name for a set of fields. All fields with the same write group name are in the same group. If users enter the write password for any member of a field group, access is granted as if the passwords were entered for all fields in the group.

10.5 Process field passwords (procpass)

After field-level security is implemented by specifying passwords, those specifications must be processed to produce a binary file that can be read by other Unify DataServer/ELS functions. Process Field Passwords compiles the password specifications to produce the binary password file named upasswd.

To process the current set of field level passwords, select Process field passwords (procpass) from the database Maintenance menu. The following screen form displays:

```
[procpass]              Unify RDBMS              30 APR 1999
                   Process Field Passwords




PROCEED? ■
```

If you enter a Y or y, the program produces the upasswd file in the current directory. The program stores upasswd in the directory specified in DBPATH if this environment variable is set (see chapter 5, Environment variables).

If you enter an N, n, or CTRL U. you exit the program and return to the menu.

10.6 Using the operating system security

The security levels described in the previous chapters are only as secure as the operating system environment. If users can run programs designed to defeat the security, or if they can dump the contents of the database files directly, they can access unauthorized data.

This chapter describes how to use the operating system to create a more secure environment for Unify DataServer/ELS. To develop a more secure environment that combines the levels of security provided by Unify DataServer/ELS with that of the operating system, consider the following:

1. The most important level of security is the Unify DataServer/ELS menu system. You must keep the Unify DataServer/ELS super user password secure, so it is used only by the system administrator.

   Enter or change this super user information using Modify system parameters & security. See Modify system parameters & security, section 10.1, for information on how to modify the user ID and password for the super user.

2. You should also assign the database design and recovery programs such as Design and create a new database, Modify database design, Write database backup, Read database backup, Transaction logging status, and Reconfigure database to another user ID. This minimizes the need to distribute the super user ID and password to other users.

3. Assign access to Add or modify group privileges and Add or modify individual privileges with care.

   For example, you can limit access to these programs to the super user.

4. Verify the procedures used in any program before the program is made known to the menu system. This involves the Describe program to system program.

   As described, you can establish operating system security in such a way that programs can access the database via Unify DataServer/ELS only.

   For example, an employee may have read privilege on the paycheck file. But this doesn't mean the same employee must be able to register a check printing program.

5. Establish a special operating system user ID to prevent access to the database by unauthorized user programs. This user ID should own all the system files for the database, such as unify.db, file.db, file.dbr, and upasswd. Only the owner should be able to read and write these files.

   Move the program Unify DataServer/ELS to its own directory, with all other Unify DataServer/ELS executables. The executables must be owned by this special user. Then turn on the set user ID bit.

   For more information, see chmod in your operating system Programmer's Manual.

   NOTE: Unify DataServer/ELS creates files using its own permission modes. Any use of umask in custom programs is ignored. If you want to prevent other users from using your database, you must either change the mode of the directory where your database resides, or change the mode of the individual database files.

For added operating system level security, the noper program prints the following message if a user who does not have shell escape permission tries to run sh or edit from ENTER, SSQL, or SQL:

Permission denied.

The noper program is located in the /unify/bin directory.

Chapter 11: Data Dictionary reports

This chapter describes the standard Unify DataServer/ELS data dictionary reports and three database statistics reports.

The data dictionary reports display as options on the Data Dictionary reports menu. These options let you list the database design, menus, screens, group privileges, individual privileges, help documentation, and programs used in your database application.

Because the data dictionary is a Unify DataServer/ELS database, you can query the data dictionary like any other database for the information you want. However, the Data Dictionary reports list the information you probably want to see most often.

The other three reports described in this chapter list database statistics, hash table statistics, and B-tree statistics.

11.1 Print database design(schlist)

The Database design report is actually six reports: Schema Listing, Table list, Field list, Table relationships, Table change list, and Field change list.

The Database Design report lists tables and fields in the order they were created. Under each table name it lists that table's unique short field names and descriptive long field names. The report does not separate fields that refer to COMB type fields in other tables into their component fields.

The Table List reports the lengths of tables in bytes including overhead needed by Unify DataServer/ELS.

The Field list report lists all fields in the database in alphabetical order. This report does separate fields that refer to COMB type fields into their component fields by listing the implicit names of the component fields. The Field list report also lists a field number for each field. This number refers to the field's internal position in the database.

To print the database Design reports, select the Print database design (schlist) option from either the database design utilities or Data Dictionary reports menu. This program has no options to select or prompts to answer. The Data Dictionary reports print in the order shown in figures 11.1 through 11.6. The reports are 79 columns wide, but have been reduced for this document.

```
DATE: 04/30/99  TIME: 11:29:35                          PAGE: 1
DATA DICATIONARY REPORT
Schema Listing
TABLE/FIELD     REF             TYPE        LENGTH   LONG NAME
-------------------------------------------------------------------
MANF            10                                   manufacturer
    *mano                       NUMERIC     4        Manufacturer_ID
     mname                      STRING      35       Name
      madd                      STRING      30       Address
      mcity                     STRING      20       City
```

Figure 11.1 Database design report

DATE: 04/30/99 TIME: 11:29:35 PAGE: 1

DATA DICATIONARY REPORT

Schema Listing

TABLE/FIELD	REF	TYPE	LENGTH	LONG NAME
mstate		STRING	2	State
mzip		STRING	5	Zip_Code
model	50			model
*mokey		COMB		Model_Key
monum		NUMERIC	7	Model_Number
momano	mono	NUMERIC	4	Manufacturer_ID
mdes		STRING	30	Description
item	100			inventory_item
*sno		NUMERIC	9	Serial_Number
imodel	mokey	COMB		Model_Key
iad		DATE		Acquisition_Date
isal		AMOUNT	5	Sales_Price
iorder	onum	NUMERIC	9	Order_Number
ipamt		AMOUNT	5	Purchase_Price
customer	10			customer
*cnum		NUMERIC	5	Customer_Number
cname		STRING	30	Name
caddr		STRING	30	Address
ccity		STRING	20	City
cstate		STRING	2	State
czip		STRING	5	Zip_Code
cphone		STRING	14	Phone_Number
orders	100			order

Figure 11.1 Database design report

```
DATE: 04/30/99  TIME: 11:29:35                          PAGE: 1
DATA DICATIONARY REPORT
Schema Listing
TABLE/FIELD     REF             TYPE        LENGTH   LONG NAME
-------------------------------------------------------------------
     *onum                      NUMERIC     9        Order_Number
      odate                     DATE                 Date_Ordered
      ocust              cnum   NUMERIC     5        Customer_Number
```

Figure 11.1 Database design report

The column headings on the Schema Listing Report are described as follows:

TABLE/FIELD
: The short names of each table in the database and the fields in each table, listed in the order in which they were defined.

REF
: The names of each field referenced in an explicit relationship.

TYPE
: The data types of the fields in each table.

LENGTH
: The display lengths of the fields in each table.

LONG NAME
: The long name of each database table and field.

```
DATE:04/30/99   TIME:11:29:46                PAGE: 2
DATA DICTIONARY REPORT
Table List
TABLE           NUMER           EXPECTED    LENGTH
-------------------------------------------------------
customer        4               10          122
item            3               100         52
manf            1               10          112
model           2               50          64
order           5               100         38
```

Figure 11.2 Table list report

The column headings on the table list report are described as follows:

TABLE The short name of each table in the database, listed in alphabetical order.

NUMBER The number that identifies each table in the data dictionary.

EXPECTED The number of records each table is expected to contain.

LENGTH The record length in bytes for each table..

DATE: 04/30/99 TIME: 11:29:49 PAGE 3

DATA DICTIONARY REPORT

Field List

FIELD	NUMBER	TABLE	TYPE	LENGTH
caddr	21	customer	STRING	30
ccity	22	customer	STRING	20
cname	20	customer	STRING	30
cnum	19	customer	NUMERIC	5
cphone	25	customer	STRING	14
cstate	23	customer	STRING	2
czip	24	customer	STRING	5
iad	12	item	DATE	
imodel	9	item	COMB	6
imodel_momano	11	item	NUMERIC	4
imodel_monum	10	item	NUMERIC	7
iorder	17	item	NUMERIC	9
ipamt	18	item	AMOUNT	5
isal	13	item	AMOUNT	5
madd	3	manf	STRING	30
mano	1	manf	NUMERIC	4
mcity	14	manf	STRING	20
mdes	7	model	STRING	30
mname	2	manf	STRING	35
mokey	4	model	COMB	6

Figure 11.3 Field list report

DATE: 04/30/99 TIME: 11:29:49 PAGE 3

DATA DICTIONARY REPORT

Field List

FIELD	NUMBER	TABLE	TYPE	LENGTH
momano	6	model	NUMERIC	4
monum	5	model	NUMERIC	7
mstate	15	manf	STRING	2
mzip	16	manf	STRING	5
ocust	28	orders	NUMERIC	5
odate	27	orders	DATE	
onum	26	orders	NUMERIC	9
sno	8	item	NUMERIC	9

Figure 11.3 Field list report

The column headings on the Field list report are described as follows:

FIELD — The short name of each field in the database (including the access name for each component field of a COMB field), listed in alphabetical order.

NUMBER — The number that identifies each field in the data dictionary.

TABLE — The short name of the table that contains the field.

TYPE — The data type of each field.

LENGTH — The length of each field..

DATE: 04/30/99 TIME: 11:29:49 PAGE 4

DATA DICTIONARY REPORT

Table Relationships

TABLE	TABLE	FIELD
item	model	imodel
item	manf	imodel_momano

Figure 11.4 Table relationships report

```
DATE: 04/30/99  TIME: 11:29:49                    PAGE 4
DATA DICTIONARY REPORT
Table Relationships
TABLE           TABLE           FIELD
item            orders          iorder
model           manf            momano
orders          customer        ocust
```

Figure 11.4 Table relationships report

TABLE — The short name of each *child* table that contains an explicit relationship.

TABLE — The short name of the *parent* table that contains the field referenced by each *child* table.

FIELD — The short name of the field in each *child* table that references a field in the *parent* table. If the field is a component of a COMB type field, the access name of the component field is listed.

```
DATE: 04/30/99 TIME: 11:29:49              PAGE: 5
DATA DICTIONARY REPORT
Table Change List
TABLE           CHANGE
orders          A
model           M
tester          D
```

Figure 11.5 Table change list report

The column headings on the Table Change List Report are described as follows:

TABLE — The short name of each table that has been changed.

CHANGE — A single character indicating the change that has been made to the table:

A — The table has been added to the database

M The table has been modified

D The table has been deleted from the database.

```
DATE: 04/30/99   TIME: 11:29:49                    PAGE: 6
DATA DICTIONARY REPORT
Field Change List
FIELD            CHANGE
-------------------------------------------------------------
madd             M
mcode            D
ostat            D
```

Figure 11.6 Field change list report

The column headings on the Field Change List Report are described as follows:

FIELD The short name of each field that has been changed.

CHANGE A single character indicating the change that has been made to the field:

A The field has been added to the table

M The field has been modified

D The field has been deleted from the table

11.2 Print menus (lmenu)

The Print menus program produces a report of all menus in the database's data dictionary. The report is 79 columns wide, and lists menu names and headings by name, with menu selection options by number, beneath each menu title.

To print the report, select Print Menus (lmenu) from the Data Dictionary reports menu. This program has no options to select.

DATE:04/30/99 TIME:11:29:46 PAGE: 6
DATA DICTIONARY REPORT
Print Menus

MENU	HEADING/PROMPT	MENU/PROG	M/P
entmenu	Unify Main Menu		
1	File Maintenance Menu	fmntmen	M
2	Report Menu	rptmenu	M
3	SQL _ Query/DML Language	sql	P
4	Create or Modify Screen Forms	screens	M
fmntmen	File Maintenance Menu		
1	Manufacturer Maintenance	sman100	E
2	Model Maintenance	smod100	E
3	Item Maintenance	sitm100	E
rptmenu	Report Menu		
1	Inventory Item Overview	invitmo	P
2	Inventory Aging Report	aging	P
3	Purchase Acquisitions	acqlist	S

Figure 11.7 Menu list report

```
DATE:04/30/99   TIME:11:29:46                                  PAGE: 6
DATA DICTIONARY REPORT
Print Menus
```

MENU	HEADING/PROMPT	MENU/PROG	M/P
screen	Create or Modify Screen Forms		
1	Paint Screen Forms	paint	P
2	Register Screen Form with ENTER	entmnt	P
3	Register Screen Form with SQL	ssqlmnt	P
4	Check Screen Form Coordinates	sfmaint	P
5	Display List of Screens	sfslist	P
6	Test Screens	sfsamp	P
7	Compile Screens	sfproc	P
8	Restore Screen to Data Dictionary	sfrestr	P
9	Create Default Screen Form	cdsf	P

```
TOTAL MENUS: 4
```

Figure 11.7 Menu list report

The column heading on the Print menus report are described as follows:

MENU The short system name of each menu, and the numbers of the options on the menu.

HEADING / PROMPT
The heading that displays on the second line of the screen when each menu is displayed, and the prompts that display for each menu's options.

MENU/PROG M/P
The short system names of the options that display on each menu.

M/P A single character that indicates each option's type:

M	The option is another menu
P	The option is a program
E	The option is an ENTER screen
S	The option is an SQL screen

11.3 Print screens (sfrep)

The Print screens report prints a copy of the selected screen form, a list of all the screen field information, or both. This information can help you document an application system. The report is 120 columns wide.

To generate this report, select Print screens (sfrep) from the Data Dictionary reports menu. The following screen form displays:

```
 [sfrep]              Unify RDBMS                30 APR 1999
                    Print Screens


SCREEN          :

LISTING (Y or N):
PRINT SCREEN:
```

The prompts on this screen are described as follows:

SCREEN — If you want to print a specific screen form, enter its name at this prompt. If you want to print all forms in the data dictionary, enter ALL.

LISTING (Y or N)
: If you want to print a tabular report of the attributes for each screen field, answer Y at this prompt. If you want to omit this part of the report, answer N.

PRINT SCREEN
: If you want to print an image of the screen form as it displays on a CRT, answer Y at this prompt. If you want to omit the screen form image, answer N.

11.4 Print group privileges (lgrp)

The Print group privileges option prints the Group access listing. This report lists all user groups in the data dictionary by group ID. Following each group is a list of the menus and programs that the group is allowed to access and the access privileges for the programs. The report is 79 columns wide.

To print the Group Access Listing report, select Print group privileges (lgrp) from the Security maintenance menu. This program has no options for you to select.

```
DATE: 04/30/99    TIME: 11:29:46                    PAGE: 1
DATA DICTIONARY REPORT
Group Access Listing
                                                ACCESS PRIVILEGES
ID    NAME         ENTRY PT    MENU/PROG       INQ   ADD   MOD   DEL
DEC   Data Entry   fmntmen     sman100     E   y     y     y     n
      Clerk
                               smod100     E   y     y     y     n
                               sitm100     E   y     y     y     n
```

Figure 11.8 Group access listing report

The column headings on the Print group privileges report are described as follows:

ID — The code that identifies each group that has access to the Unify DataServer/ELS system.

NAME — The user group name that further identifies and documents the group.

ENTRY PT — The short name of the initial menu the group sees on login to Unify DataServer/ELS.

MENU/PROG — The name of each menu, program, or screen that the group can access. Each name is followed by a single character that indicates the option's type:

M A menu

P A program

E An ENTER screen

S An SQL screen

INQ ADD MOD DEL

A single character in each of these columns indicates whether the group has that access mode (INQUIRE, ADD, MODIFY, or DELETE) for each menu, program, or screen.

y The group can use that access mode with the menu, program, or screen.

n The group cannot use that access mode with the menu, program, or screen.

11.5 Individual access listing (lemp)

The Individual access listing report lists all the users in the data dictionary by user login ID. After each individual's name, the report lists the programs that the user can access differently from other users in the group.

To print this report, select Print individual privileges (lemp) from the menu. This program has no options for you to select. The report is 132 characters wide, but its size has been reduced for this manual.

```
DATE: 04/30/99   TIME: 11:29:46                          PAGE: 1
DATA DICTIONARY REPORT
Print Individual Privileges
                                        SPECIAL ACCESS PRIVILEGES
ID   NAME          GROUP    ENTRY     MENU/PROG       INQ  ADD  MOD  DEL
jhd  John H, Doe   DEC      fmntmen
sss  Sam S.        DEC      entmenu   sman100     E   y    y    y    y
     Spade
                                      smod100     E   y    y    y    y
                                      sitm100     E   y    y    y    y
                                      sql         P   y
                                      man3000     P   y
                                      fmntmen     M   y
                                      acqlist     S   y
```

Figure 11.9 Individual access listing

The column headings on the Individual access listing report are described as follows:

ID — A code that identifies an individual who has access to the Unify DataServer/ELS system.

NAME — The 30-character name that further identifies and documents the individual.

GROUP — The code that identifies the group to which the individual belongs.

ENTRY PT The short name of the initial menu the individual sees on login to Unify DataServer/ELS.

MENU/PROG

The names of the menus, programs, or screens that the individual can access. Each name is followed by a single character that indicates whether it is a menu, program, or screen:

M	A menu
P	A program
E	An ENTER screen
S	An SQL screen

ACC INQ ADD MOD DEL

The access modes the individual has for each menu, program, or screen. A y or an n in the ACC column indicates whether the individual has access to the option. A y or an n in the INQ, ADD, MOD, or DEL column indicates whether the individual has inquire, add, modify, or delete privileges for the option.

11.6 Print help documentation (prthdoc)

The Print help documentation program prints the help text associated with a menu option.

To print the report, select Print help documentation (prthdoc) from the Data Dictionary reports menu. Figure 11.10 shows an example of the help documentation report.

```
                    Inventory Item Overview

This report lists items in inventory, grouped by description and
manufacturer. There is a group count of each identical item in
stock made by the same manufacturer. The report also gives the
average sales price of the item grouping.
```

Figure 11.10 Help documentation report

You can also display help documentation on the screen by entering help and the menu option name at the SELECTION: prompt.

11.7 Print list of programs (lexec)

The Print list of programs option prints the Executable listing report. This report lists all the executables in the data dictionary, as well as the programs in each executable.

To run this report, select print list of programs (lexec) from the Data Dictionary reports menu. This program has no options for you to select. The report is 93 columns wide, but has been reduced for this document.

```
DATE: 04/30/99      TIME: 11:29:46          PAGE: 1
DATA DICTIONARY REPORT
Executable Listing
EXECUTABLE  SYSFLAG  PROGRAM  HEADING          IDX  SCREEN       DIRECTORY
-------------------------------------------------------------------------
ENTER                smod100  Model            0    smod100
                              Maintenance
                     sman100  Manufacturer     0    sman100
                              Maintenance
HTS         N        hts      Display Hash     0
                              Table
                              Statistics
IDXMNT      N        idxmnt   Add, Drop        0
                              B_Tree
                              Indexes
LST         N        1st      Listing          0
                              Processor
MOD300      N        mod300   Model Listing    0
ORD100      Y        ord100   Order            0    sord100 ord
                              Maintenance
SQL         N        sql      SQL-Query/DML    0
                              Language
```

Figure 11.11 Executable listing report

The column headings on the Executable listing report are described as follows:

EXECUTABLE
: The name of each file that contains an executable registered with the Unify DataServer/ELS system.

SYSFLAG
: A single character that indicates whether the executable uses the sysrecev function:

  y
  : A standard argument list will be passed to the executable from the Menu Handler when the executable is started. The argument list is interpreted by sysrecev.

  n
  : No arguments will be passed to the executable from the Menu Handler when the executable is started.

PROGRAM
: The name of each program in the executable file.

HEADING
: The heading that displays on line two of the screen when the program is running.

IDX
: The index code of each program in the executable. If the executable contains only one program, that program's index code is zero (0). If the executable contains several programs, the programs' index codes are zero (0) to one less than the number of programs.

SCREEN
: The name of an optional screen form that sysrecev displays before the program is started.

DIRECTORY
: For a custom program, the name of the directory where the source code is located.

11.8 Print, display database statistics (dbstats)

The database statistics report lists statistics for various database parameters.

To run this report, select Print, display database statistics (dbstats) from the database maintenance menu. The following prompt lets you select the output destination for the report:

[P]rint or [D]isplay Statistics?

If you want to send the report to the printer, answer P at the prompt. If you want to display the report on the screen, answer D. The database statistics report is 79 characters wide, and has been reduced for this document.

```
DATE: 04/30/99  TIME: 11:29:46                          PAGE: 1
DATA BASE REPORT
Data Base Statistics
TABLE           ADJUSTED          TOTAL
NAME      USED  EXPECTED    %     ALLOCATED  %     DELETED  AVAILABLE
manf      6     12          50.0  20         30.0  0        14
model     15    54          27.8  90         16.7  0        75
item      14    102         13.7  180        7.7   0        166
VOL# TYP  CHAR  DEV BLOCKED DEV   SIZE       MAX
0    F                            39                       (file.db)
TOTAL DATA BASE SIZE              39
```

Figure 11.12 Database statistics report

The column headings on the database statistics report are described as follows:

TABLE NAME
: The name of the table.

USED
: The total number of slots that have been used, including the spaces formerly occupied by deleted records.

ADJUSTED EXPECTED
: The approximate number of expected records. The space allocation algorithm adjusts the number of records to fit in whole blocks. Therefore, this number may not be exactly the expected number entered through Design and create a new database or Modify database design.

(%)
: The percent used of adjusted expected record space.

TOTAL ALLOCATED
: The number of records for which space is allocated. This allows room for overflow.

  A given table can contain 166% of the expected number of records. Then more space must be allocated by modifying the database design and reconfiguring the database.

(%)
: The percent used of the total space allocated for records.

DELETED
: The number of records marked for deletion.

AVAILABLE
: The amount of available records space:

  AVAILABLE =
  TOTAL ALLOCATED - USED + DELETED.

VOL #
: The volume number. Volumes are numbered 0 through 7, beginning with the root volume which is always 0.

TYP
: The volume type. This is either F for file or D for a disk or tape device.

CHAR DEV
: For TYP F, the system name of the ordinary file for the database. For TYP D, the system name of the character or block special file for the database.

BLOCKED DEV
: For TYP F, the name of the ordinary database file. For TYP D, the system name of the block special file for the database.

SIZE
: The size of the volume in 512-byte blocks.

MAX
: The maximum size of the volume in 512-byte blocks, specified by the user.

TOTAL database SIZE
: The current size of the database in 512-byte blocks.

11.9 Display hash table statistics (hts)

The Hash table statistics report displays statistics for various hash table parameters. To display this report on the screen, select Display hash table statistics (hts) from the database maintenance menu. The program has no options for you to select.

The following is an example Hash table statistics report.

```
[hts]              Unify RDBMS                  30 APR 1999
                  Display Hash Table


                HASH TABLE STATISTICS


  Total Entries                107601
  Per Cent Filled                  42.00
  Average Chain Length              3.08
  Longest Chain Length            51
  Starting Address               56896

                Number of Chains of Length

 2     3      4    5    6    7    8    9    10    >10
 3568 14165   18    0    0    97   466  71   103    1613

  ->-> ■
```

When you are finished looking at the report, press RETURN.

The items on the report are described as follows:

Total Entries The total number of filled slots in the hash table.

Per Cent Filled
The percent of entries filled. If 50% of the slots are filled, the table is at

optimal density. If 70% or more of the slots are filled, the performance of acckey and addrec degrades.

Average Chain Length
: The average number of consecutive filled slots before an empty slot in the table. 50% of this number is a gross measure of the probes necessary to find a record.

  In the example, an average of 1.5 probes are necessary to find a record. The actual number can be less, because the table is stored in the hash table, and consecutive slots might be filled with entries from a different table. This number should be 6 or less.

Longest Chain Length
: The longest sequence of consecutive slots that are filled. This gives a gross measure of the worst case access time.

Starting Address
: The address of the first slot of the longest chain. This is where you look in the hash table if you want to see what the longest chain looks like.

Number of Chains of Length...
: The number of sequences of consecutive filled slots of each length. This shows the distribution of chain lengths.

You can use hash table statistics to diagnose overall system performance. A decline in system performance may be caused by a full database or clustering of hash table entries.

Database performance is best when Per cent filled is 75% or less. Likewise, the best performance is given when chains are length 6 or less and where the average chain length is under 2.5; higher numbers indicate that clustering is occurring.

If the hash table statistics exceed these ranges, run the rekey program; then check the statistics again. If the problem has not been corrected, you can improve performance by increasing the size of the hash table.

To increase the hash table size, increase the expected number of records. Or, create an empty table with a larger number of expected records.

11.10 Print, display B-tree statistics (btstats)

The B-tree statistics report lists all B-tree indexes, sorted by table and index ID number. Each section of the report lists the index file name, size, database fields, sort order, ID number, and whether duplicates are allowed. The report is 79 columns wide.

To run this report, select "Print, display B-tree statistics" (btstats) from the database maintenance menu.

```
DATE: 04/30/99  TIME: 11:29:46                    PAGE: 1
Database REPORT
B-Tree Statistics
Table #         1:  manf    Record count = 10

   Index ID Number:       1     - Duplicates are NOT allowed.
   Index file name: btOOl.idx
   Index file size:       728 bytes

   Order    Field Name
     A       mnumber
     A       mstate

   Index ID Number:       2       - Duplicates are allowed.
   Index file name:bt004.idx
   Index file size:       0 bytes  * Index file is EMPTY
                                    *Index must be REBUILT

   Order    Field Name
     A       mstate
     D        mzip
```

Figure 11.13 B-tree statistics report

The prompt [P]rint or [D]isplay Statistics? lets you choose whether you want to send the report to the printer or to the screen. If you want to send the report to the printer, answer P at this prompt. If you want to display the report on the screen, answer D.

The headings that display on the B-tree statistics report are described as follows:

Table # The number in the data dictionary that identifies the table containing the B-tree indexes. The table number is followed by the table's short name.

Record count
The number of records the table is expected to contain.

Index ID Number:
The number that identifies the B-tree index. This is the number you enter to modify or delete a B-tree index in Add, Drop B-Tree Indexes.

Index file name:
The system name of the file that contains the B-tree index.

Index file size:
The size in bytes of the index file.

Order A single character that indicates the order in which each field's values are sorted in the B-tree index:

A	Ascending order
D	Descending order

Field Name The name of each field in the B-tree index. The report may display additional information about each B-tree. This information includes whether duplicates are allowed in the index, whether the index file is empty, and whether the index needs rebuilt.

Chapter 12: Miscellaneous utilities

This chapter describes programs you can use to manage transactions, menus, help documentation, and executables in the data dictionary. It also tells you how to load binary executable files using the standard Unify DataServer/ELS load macro, how to log transactions, and how to use the Unify DataServer/ELS editing features.

12.1 Transaction logging

This chapter describes how transaction logging works in Unify DataServer/ELS, and how to set and display transaction logging parameters.

How transaction logging works

Transaction logging is a way to restore the database to a logically consistent state, with a minimal loss of information, after a hardware or software failure. Transaction Logging tracks updates at the function level to ensure physical database integrity. It writes a log record when the database is updated by adding or deleting a record, or modifying a field.

However, the user application program must maintain logical database integrity. Only the application recognizes that a series of database functions should be grouped in a single logical transaction.

The application program uses the startltg and endltg functions to indicate that a set of database updates must be treated as a single logical transaction. For more information on these functions, see the *Unify DataServer/ELS HLI Programmer's Manual.*

The Unify DataServer/ELS programs that update the database, ENTER, SQL, and DBLOAD, issue calls to mark the start and end of what they consider a logical transaction.

ENTER treats a set of updates made to a record as a logical transaction. SQL treats all updates made by one DML statement as a logical transaction. And, DBLOAD treats an entire set of additions or updates as a logical transaction.

If you have enabled transaction logging using the Transaction logging status program, each update made to the database is logged to the log file. For more information, see Transaction logging status in this chapter.

When a software or hardware failure occurs, any transactions still in progress cannot be completed. In the transaction log, each incomplete transaction has a "start transaction" entry with no corresponding "end transaction" entry.

However, the completed transactions in the transaction log contain all the updates that have been made, except those in incomplete transactions.

If you have had a hardware or software failure and you suspect the database has been damaged, you can use *roll forward recovery* to restore logical database integrity with a minimal loss of information.

The most common method of roll forward recovery is to use the Read Database Backup program (see Read database backup in Database maintenance utilities). This program guides you through the following steps to recover the database:

1. Read in the latest logically consistent copy of the database. Note that this copy may be several hours or even days old.
2. Read in the transaction log to find the IDs of the incomplete transactions.
3. Read in the transaction log again, to update the database with completed transactions.
4. Manually re-enter the updates lost because they were incomplete transactions.

Another method of roll forward recovery is to restore the updates from the transaction log using the replay program. The replay program is run from the shell. Use the following command syntax to run the program:

REPLAY [-c] [-n] [-b] [-v] [-e *error file*] {-i *incomplete file*]

where the following are the replay parameters:

-c Check the transaction log. This is an optional parameter to check the transaction log for incomplete logical transaction groups without replaying transactions. Running replay with this parameter creates the incomplete transaction group summary file (described under the -i option) and the date of the last valid transaction group. The incomplete summary file contains the comments from and number of each incomplete group. The incomplete summary file is also created when tranactions are replayed.

-n Omit the incomplete transaction group report.

Use this parameter to check the transaction log or replay transactions without creating the incomplete transaction group report.

-b Read in the transaction log from the backup device.

Use this parameter to read the transaction log from the backup device specified by the environment variable BUDEV, instead of from the standard transaction logging device.

-v List every transaction in the incomplete groups (verbose list).

Use this parameter to list each group's incomplete transactions in the incomplete transaction group summary file.

-e Name the error file.

Use this option, followed by an *error file* name, to change the name of the error file. The default name is replay.err.

-i Name the incomplete transaction group summary file.

Use this option, followed by the new *incomplete file* name, to change the name of the file used for the incomplete transaction group report. The default name is replay.not.

Transaction logging status (txconf)

The Transaction logging status (txconf) program enables you to set and display the current transaction logging parameters. You can set transaction logging ON or OFF, name a log file that records all transactions, specify the size of the log file, and name a file that records transaction logging errors.

To run this program, select Transaction logging status (txconf) from the System Administration menu. The following screen form displays:

```
[txconf]              Unify RDBMS                30 apr 1999
                Transaction Logging Status


CURRENT LOGGING STATUS              :off

REQUESTED LOGGING STATUS (on/off)   :off
- — — — — — — — — — — — — — — — — — — — — — — — — — — — — -

 NAME OF LOG FILE (OR DEVICE NAME)    :  txlogfile

 TYPE OF LOG FILE (file/device)       :  file

 IS LOGGING MEDIA REMOVABLE?          :  no

 SIZE OF LOG FILE (512b block)        :  4000

 NAME OF ERROR REPORTING FILE         :  txerrorfile
```

The prompts are described as follows:

CURRENT LOGGING STATUS:

A display-only field that shows the current status of transaction logging. If the transaction logging status is ON, database updates are recorded in the transaction log. If the transaction logging status is OFF, database updates are not recorded in the transaction log.

REQUESTED LOGGING STATUS (on/off):

This field lets you set the transaction logging status.

Enter OFF to set the flag to OFF and stop transaction logging. (The default is OFF.)

Enter ON to set the flag back to ON and process transaction logging only if the database has not been changed since the last backup. If the database has been changed, logging starts only after the database has been backed up.

NAME OF LOG FILE (OR DEVICE NAME):

The name of the file or device name that records all transactions while transaction logging is on. The default name is txlogfile.

You can specify a log file or device name in one of three ways:

1. By a file name. In this case, the file is located in the directory set by DBPATH, or in the current directory if DBPATH is not set.
2. By a complete path name, beginning with a /. In this case, the file is located exactly where you have specified.
3. By a device name. In this case, the name must refer to an ordinary (blocked) disk or diskette device; it cannot refer to a tape device.

TYPE OF LOG FILE (file/device):

A keyword you enter to specify whether the transaction log is an ASCII file or a hardware device.

You can enter either the file or device keyword. The default setting is file.

IS LOGGING MEDIA REMOVABLE?:

Only applicable if the transaction log is a hardware device. This field indicates whether the media is removable.

If the media is removable and you want to be prompted when to mount and unmount the transaction log, enter y (yes) at this prompt. The default is n (no).

SIZE OF LOG FILE (512b blocks):

The maximum size of the transaction log, in 512-byte blocks. The default size is 4000 blocks.

The log file size limit is useful if your transaction log is a removable diskette and you want transaction logging to stop before the log is full.

ENTER reports when the log is 50, 80, 90, and 100% full. When the log is 100% full, transaction logging is turned off.

NAME OF ERROR REPORTING FILE:

The name or path of the ASCII file that stores error messages generated by transaction logging. For example, if a write error occurs while attempting to write an entry in the transaction log, transaction logging sends a message to this file.

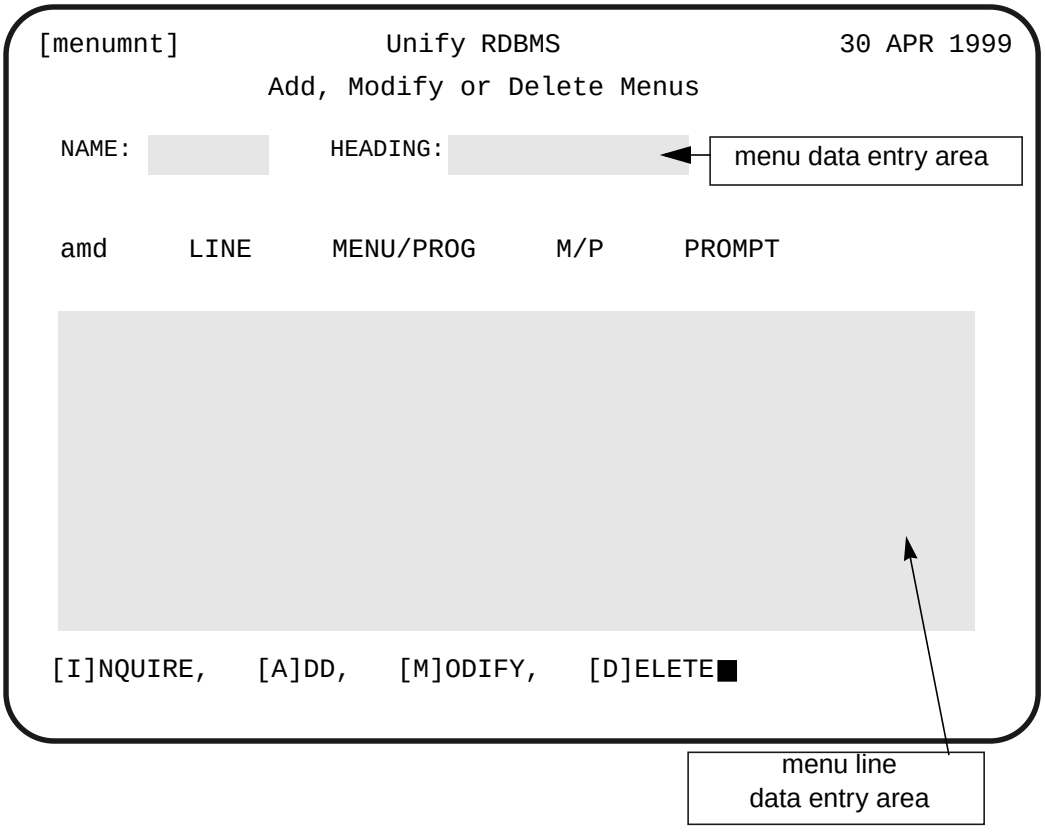
The default name is txerrorfile. This file is opened in the $DBPATH directory; if DBPATH is undefined, the file is opened in the current directory.

If the error reporting file cannot be written to, or cannot be opened by the error routines, transaction logging errors display on the screen.

NOTE: After you have changed transaction logging from OFF to ON, you must run Write database backup to activate a new log.

12.2 Add, modify or delete menus (menumnt)

To start this program, select Add, modify or delete menus (menumnt) from the main menu. The screen form displays as follows:



The Add, modify or delete menus screen form enables you to inquire about, add, modify, and delete menus and menu lines in the data dictionary.

The screen form has a menu data entry area where you enter the menu name and heading and a menu line data entry area where you add, modify, and delete menu lines.

Each menu can have up to 16 lines in the menu line data entry area. This is because the data dictionary is limited to 500 menu lines and 16 menu lines per menu.

The prompts and column headings on the screen form are described as follows:

[I]NQUIRE, [A]DD, [M]ODIFY, [D]ELETE

The operation prompt. You can select one of the following operation modes at this prompt:

i	Inquire mode. This lets you view an existing menu.
a	Add mode. This lets you create a menu.
m	Modify mode. This lets you add, modify, and delete lines on the menu.
d	Delete mode. This lets you delete a menu.

NAME

A unique menu name of up to eight characters. It must be different from other menu, screen form, and program names.

HEADING

A string of up to 34 characters that displays on the second line of the screen form when the menu is active and on other menus where this menu is an option.

amd

The column where you enter an operation command for the current menu line. The cursor marks the current menu line. You can move the cursor up or down in the menu line data entry area by pressing CTRL U or RETURN. The valid operation commands are as follows:

a	Add a line to the menu. This response is valid only on an empty line in the menu line data entry area.
m	Modify LINE or MENU/PROG for the current menu line.
d	Delete the current menu line.
q	Leave the menu line data entry area and position the cursor at the HEADING prompt.

LINE — The position of the current menu line on the menu. You can enter any number between one and 16. The menu lines will be resorted in the new sequence when you enter CTRL U from this column.

MENU/PROG — The name of the menu, program, ENTER screen, or SQL screen that starts when you select this menu line.

M/P — A display-only field that indicates the type of the MENU/PROG entry. The characters that can appear in this column are defined as follows:

M	A menu
P	A program
E	An ENTER screen
S	An SSQL screen

PROMPT — A display-only field that indicates what text displays on the menu for the current menu line. The table in figure 12.1 shows where the PROMPT comes from.

When the current entry is...	Then prompt comes from...
a menu	Add, modify or delete menus (this Program)
a program	Describe program to system (execmnt)
an ENTER screen	Register screen form with ENTER (entmnt)
SSQL screen	Register screen form with SQL (ssqimnt)

Figure 12.1 Sources of menu prompt

12.3 Describe program to system (execmnt)

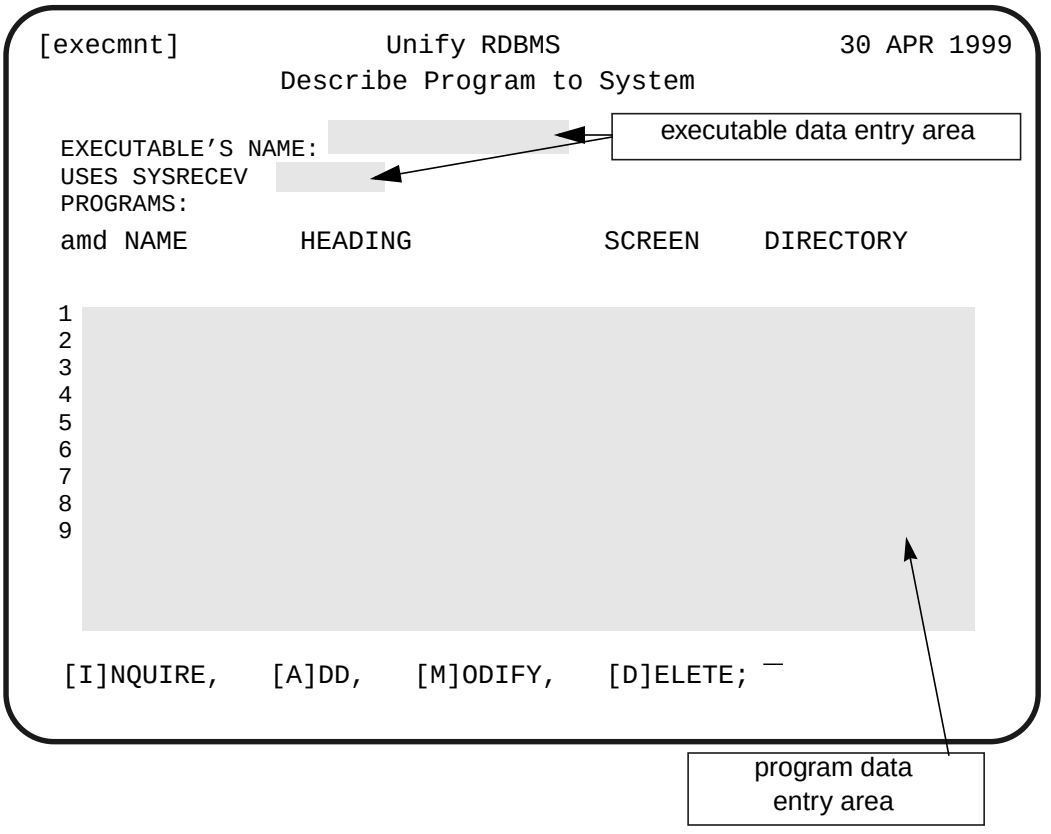
This program lets you inquire about, add, modify, and delete executable programs, and programs within executable programs. Executable programs are usually referred to as executables in Unify DataServer/ELS.

Executables can contain one or more programs. For example, some executables, such as Display hash table statistics (hts), are single programs. They each perform a single task.

Other executables may perform several tasks. These executables are often custom programs that consist of several procedures or programs. Each procedure or program is called by the main program in the executable.

Before asking UNIFY to run your executables, you must describe each executable and its programs to the Unify DataServer/ELS system, using Describe program to system.

To start this program, select Describe program to system (execmnt) from the System administration menu. The following screen form displays:



The prompts on the screen form are described as follows:

[IN]QUIRE, [A]DD, [M]ODIFY, [D]ELETE

The operation prompt. The following are the operation modes you can select:

i	Inquire mode. This lets you view information about an executable registered with the Menu Handler.
a	Add mode. This lets you register an executable with the Menu Handler.

m — Modify mode.
This lets you modify HEADING, SCREEN, and DIRECTORY for programs in an executable; and add, modify, and delete programs

d — Delete mode.
This lets you delete the an executable and its associated programs.

EXECUTABLE'S NAME

A string of up to eight characters that is the name of the file containing the executable. This file name is passed to the shell when you select a program in the executable.

USES SYSRECEV ?

A Y answer tells the Menu Handler to pass the standard argument list to the executable (see the *Unify DataServer/ELS Programmer's Manual* for a description of the arguments) -

An N tells the Menu Handler to pass no arguments to this executable. The sysrecev function usually interprets this argument list, but any program or shell script can request that these arguments be passed.

amd — The column where you enter an operation command for the current program. The cursor marks the current program. You can move the cursor up or down in the program data entry area of the screen form by pressing CTRL U or RETURN.

The operation commands are as follows:

a — Add a program to the executable. This response is valid only on an empty line in the program data entry area.

m — Modify HEADING, SCREEN, or DIRECTORY for the current program. When you want to modify NAME, you must delete the line and add the name again.

d — Delete the current program.

q — Leave the program data entry area and position the cursor at the USES SYSRECEV prompt.

NAME — A string of up to eight characters that is the system name for a program in the executable. You can enter this name at the menu SELECTION prompt to run the program.

If the executable is a custom program and uses a host language function to call this program, this name must be the same as the Host Language function name for the program. UNIFY has Host Language Interfaces for C and RM/COBOL.

HEADING

A string of up to 34 characters that displays on the second line of the screen form when the program is running, and on menus where this program is a selection option.

SCREEN

This is the name of the screen form to be displayed by sysrecev before the user program starts. This is an optional entry.

DIRECTORY

For a custom program, the name of the system directory where the program source code is located. See the *Unify DataServer/ELS HLI Programmer's Manua*l for more information.

The Unify DataServer/ELS program loading shell script (uld) expects the archive containing the program to be in this directory.

When you enter the directory name here, the relative path, .../src/, is added as a prefix to the name. If you do not enter the directory name, uld uses the first three characters of the program name as the directory name.

Suppose for example, the program name is or 100. uld looks in ../src/ ord for the archive containing ord 100.

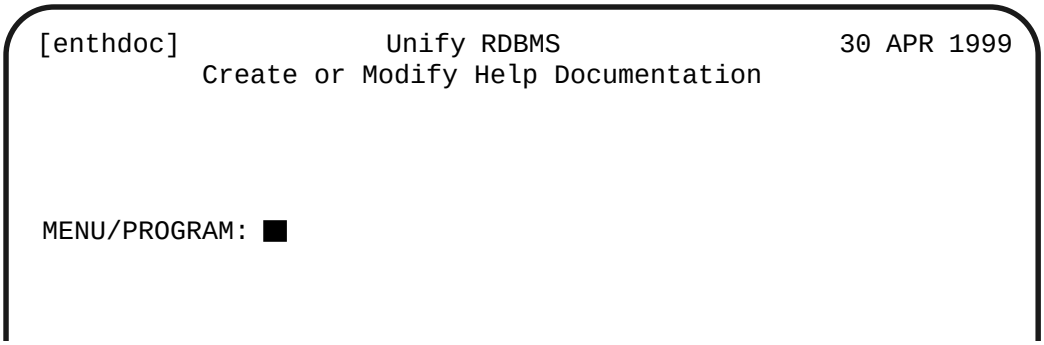
12.4 Create or modify help documentation (enthdoc)

The Create or modify help documentation program lets you develop help text to be associated with a menu option.

This help documentation displays on the screen when you enter help and the menu option name at the SELECTION: prompt. You can also send help text files to the printer using print help documentation (section 11.6).

To start this program, select Create or modify help documentation (enthdoc) from the System administration menu.

The screen form for this option displays with the following prompt:



The prompt on this screen is described as follows:

MENU/PROGRAM

Enter the system name of a menu, program, ENTER screen, or SQL screen. This system name is passed to the editor when it starts and is used for the help documentation file name. The help documentation file is named in the following way:

$DBPATH/. /hdoc/xxxx.n

where $DBPATH is the directory path specified in the DBPATH environment variable, and xxxx is the system name of the menu, program, ENTER screen, or SQL screen you specified at the MENU/PROGRAM prompt.

If a file by the specified name exists, it is opened for modification; otherwise, a new file is created.

If the EDIT environment variable has been set, the specified editor is started when you create or modify help documentation.

If the EDIT environment variable has not been set, the default editor is started. The default editor is vi.

Use standard editor commands to enter the help text file. When you finish editing the file, save it and exit the editor.

The prompt redisplays so you can enter a new menu, program, ENTER screen, or SQL screen name. CTRL U returns you to the menu.

NOTE: The help text files you enter are stored in the following directories. These directories must exist for you to create help documentation. The documentation for Unify DataServer/ELS programs and menus is stored below the system library directory—lib/hdoc.

If the Unify DataServer/ELS environment variable has been set, help documentation is stored in $UNIFY/hdoc. For more information, see chapter 5, Environment Variables.

12.5 Edit SQL or RPT command files

Start this program by selecting Edit SQL or RPT command files from the main menu, or by entering edit and the file name at the selection prompt.

- When you select this option from a menu, you start the current text editor without a file name.
- When you enter edit and a *file name* at the selection prompt, you start the current editor for that file.
- You can enter edit by itself at the selection prompt. The editor starts without a file name, letting you wait to name the file when you save it.

If the EDIT environment variable has been set, the specified editor is used. If EDIT has not been set, the default editor, vi, is used.

When you exit the editor, the active menu redisplays.

12.6 Database load (DBLOAD)

The Database load (DBLOAD) program lets you load new records and update existing records in a Unify DataServer/ELS database from an ordinary ASCII file.

If the ASCII file contains...	Then DBLOAD...
new primary key values	inserts the records
the primary key values of existing records	updates the fields in those records with the information from the ASCII file

Figure 12.2 DBLOAD updates

You can create the ASCII file to use with DBLOAD in the following ways:

1. By dumping data from an existing non-Unify DataServer/ELS application using the tools that application provides.
2. By using a text editor.
3. By using SQL (section 16.3) to dump data from an existing Unify DataServer/ELS database.

The only requirement is the format of the file must match what the load program expects. If you have a file in another format, you must edit the file to match the expected format. For example, you can use a text processing program such as sed or awk to change the format of the file.

Because ASCII files are usually created at the shell, this program is designed to run from the shell, instead of from the Menu Handler. This lets you use the full power of I/O redirection and pipes to load your data.

SQL also interfaces with this option, making it easier to load data and manipulate it in a Unify DataServer/ELS application. Refer to SQL extensions, section 16.3, for information on how to dump data from SQL to operating system files and how to load the data back to the database.

To use DBLOAD from the shell, you must make sure the PATH and Unify DataServer/ELS environment variables are set correctly. For more information on environment variables, see chapter 5.

DBLOAD expects four parameters when called from the shell. To start the program, use the following command syntax:

DBLOAD (-nl-u] [-1] [-s *separator*} [-b] *database table input-file spec-file*

For a typical database load, the optional parameters are not needed. DBLOAD will automatically insert new records and update existing records. The optional DBLOAD parameters are described as follows:

Parameter	Description
-n	New record mode. Use this parameter when you want to insert records that have no key and where you do not want to verify uniqueness. If a record key in file matches a record key in the database, DBLOAD displays a warning message about duplicate record keys; DBLOAD does not update that database record. If no key is defined for the table, DBLOAD inserts all records.
	-or
-u	Update mode. The first field of each line in file must contain a physical record address (not a key). DBLOAD updates the database record at this location with the values of the remaining fields. (See setloc in the U*nify DataServer/ELS HLI Programmer's Manual*.) This option is provided for compatibility with Release 3.2 only and is not intended for use with general applications.
-l	Don't log transactions. For best performance, use this flag to prevent entries in the transaction log.
-s	The separator. It is necessary to indicate the end of one field and the start of the next. Use the pipe, or vertical bar (|), whenever possible. Do not use a letter, a digit, or an underscore.
-b	Indicates that the input file is in binary format.

The following DBLOAD parameters are required:

database The name of the database file to load. This is either file. db, if you are loading data in the application database, or the name of your unique .db file.

table The name of the database table to load the data into.

input-file The name of the ASCII input file that contains the data to load. If this is a dash (-), the standard input is used. Thus, DBLOAD can be used at the end of a pipeline.

spec-file The name of another ASCII file that contains the list of fields in the table that DBLOAD should update. The fields are in the order listed in the input file.

You can use the default mode (both -n and -u missing) only if a key is defined for the table. This mode adds a new database record if the current key in *file* is unique, or updates the existing database record if the key is a duplicate.

The default mode requires the table to have a key. If the table does not have a key, you must use either -n or -u mode.

The specification file (*spec-file*) describes the format of the input file. The file consists of a single line with an entry for each field to be updated. You need not list every field in the table, just the ones you want to update. Nor do the fields have to be in a particular order. The order in the specification file just has to agree with the order in the input file.

The specification file has the following format:

```
field[sep]field[sep]field...
```

where *field* is the name (either the database name or the long name) of the database field, and [sep] is a single-character field separator that indicates where the field ends.

The field cannot be a COMB type field. Instead, separate COMB fields into their component subfields in both the specification and the input files.

The *separator* must not be a letter, a digit, or the underscore character (_) because these are used in field names. Choose a separator that does not occur in any STRING field. A good separator to use is the pipe, or vertical bar (|). You don't have to use the same separator between each field. But each separator in the specification file must match each separator in the input file.

The input file must have the same format as the specification file. Each record in the file consists of fields delimited by separators and ends with a newline character. DBLOAD ignores extra blanks in the field, but if STRING fields are shorter than their database design definitions, DBLOAD completes the STRING fields with blanks before loading them in the database.

Figure 12.3 illustrates the free format of specification and input files:

```
INPUT FILE

Moehr|70| clerk |6400|950.00|0.00| 5700
Colucci |40| salesrep |2200|2500.00|3000.00|6700
Amato |40| salesrep |6200|2000.00|750.00|5800
Fiorella |70| clerk |5700|800.00|0.00|68000
Brown |60| engineer |1300|6000.00|0.00|5900

SPECIFICATION FILE

Name| Dept_No | Job | Manager | Salary | Commission | Number
```

Figure 12.3 DBLOAD Input and Specification File Formats

In these examples, the record key is the last field, not the first.

If they have been defined for the fields in the input file, advanced field attributes can be used with DBLOAD, because DBLOAD also interfaces with Advanced field attributes (AFA). This lets DBLOAD validate field entries for new and existing records and set default field values for new records.

When inserting records, DBLOAD fills in AFA defaults if the fields are NOT specified in the DBLOAD specification file and they have AFA defaults defined. For more information about Advanced Field Attributes, see section 8.6.

12.7 Program loading (lfilegen)

The Program loading (lfilegen) program is used to load custom programs you have registered with the Menu Handler using Describe program to system (execmnt). Because this program only relates to C programs, it is described in the *Unify DataServer/ELS HLI Programmer's Manual.*

12.8 Monitoring locks (Impeek)

With Impeek you can monitor the number and type of locks at a specific time. You can determine if table locks are being obtained or if large numbers of locks are not being released.

To use Impeek, enter the following command at the shell prompt:

```
Impeek
```

The following figure shows an example of Impeek's output.

```
unify% Impeek
--------------------------------
trx# 8053.2 (1),      dbid:  13027.2077 rid:  1 tid: 0
[SLK]
--------------------------------
Total 22408  Busy  3156  Free:  19252
```

Figure 1. Example of Impeek output

The information displayed by Impeek can be interpreted as follows:

trx# Transaction information consisting of three parts:

- The process ID number
- The number of times this process has started a new transaction
- The number of locks held by the process now

dbid: Database information, consisting of two parts:

- The inode number of file.db
- The device number that file.db resides on.

The device number is calculated by multiplying the major device number by 256 and adding the result to the minor device number.

rid: and tid: Normally rid and tid refer to the relation (table) number and tuple (record) number that is locked. If either number is 0, it has meaning as shown in the following table:

State	Meaning
both rid and tid=0	This database is locked
only rid=0	There is a resource lock, and tid is the ASCII value for the type of lock. (See lock_r in the DBMS Direct HLI Programmer's Manual
only tid=0	There is a table lock and the rid number is the number of the table that is locked. (See Data Dictionary Report: Table List)

[SLK] or [XLK] The type lock held on this object. SLK represents shared locks and XLK represents exclusive locks.

Part IV: Screen form development and maintenance

Part IV, Screen form development and maintenance, contains chapters13 and 14.

Chapter 13, Screen form development utilities, describes the tools you can use to develop screens for data entry and inquiry. These tools include the Create Default Screen Forms program and PAINT, a full-screen editor for customizing screen forms.

Chapter 14, Screen form maintenance utilities, describes additional tools to help you manage screen forms. These tools let you check the row and column coordinates of prompts and fields on screen forms, change prompts and field positions and contents, display a list of screen forms, compile screens, and restore a damaged screen to the data dictionary.

Chapter 13: Screen form development tools

This chapter describes *screen forms* and how they are developed in Unify DataServer/ELS. A screen form is the basic tool for entering and viewing data in the database.

Paint screen forms, or PAINT, is a full-screen forms editor that makes designing and modifying screen forms quick and easy. PAINT is the major tool used to design and modify forms.

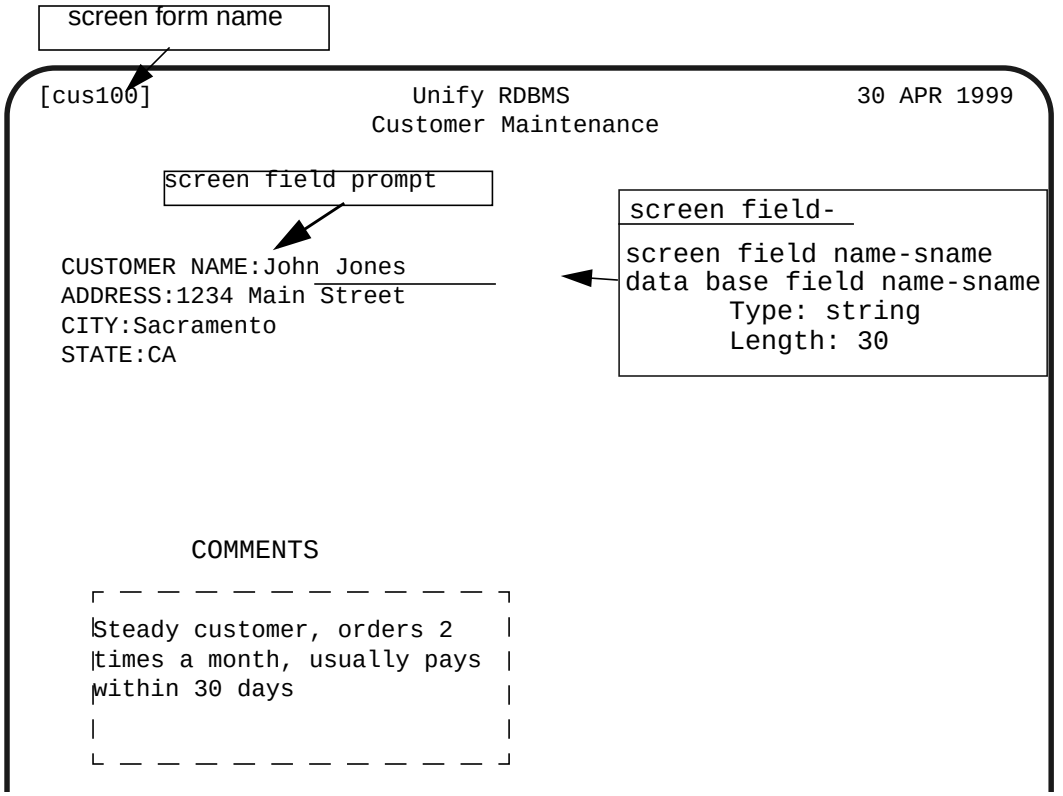
Another Unify DataServer/ELS screen form development tool is Create default screen form, described in this chapter. Create default screen form uses data dictionary information to build a default screen form for the specified table.

You can then modify the screen form using PAINT or Check screen form coordinates, described in section 14.1. Check screen form coordinates displays pages of screen form information, whereas PAINT displays fields where they appear on the screen form.

A screen form can be used in one of three ways:

1. The general-purpose data entry driver, ENTER, can be used with a screen to perform simple kinds of data entry and query applications. See chapter 15 for a description of the screen design requirements for ENTER.
2. A screen can be associated with an SQL script to let you enter parameters to be substituted in the script before it is run. This use of screen forms with SQL, called SQL by forms, is described in SQL/screen form/report interface, section 16.4.
3. For complete control over screen form processing, a set of functions can be called by custom host language programs to display or erase screen forms, and accept or display data. See the *Unify DataServer/ELS HLI Programmer's Manual* for more information.

Before you learn how screen forms are created and modified, look at the elements of a screen form. The following is a typical screen that could be used to update and view a simple customer file:.



A screen form has three elements: The *screen form name*, the *screen field prompts*, and the *screen fields*.

Every screen form must have a unique name, so it can be referred to without confusion. The screen form name in the example is cus100.

The screen field prompts are the static, unchanging, decorative elements that make up the basic screen design. In the example, CITY: and the box around the comments are screen field prompts. Screen field prompts identify, explain, or highlight *screen fields*.

Screen fields are windows into the database file. They let you see the values of database fields for the current record. In the example, screen fields let you see the values for the cname, caddr, ccity, estate, and ccomms database fields.

Each screen field consists of a *screen field name*, as well as a *database field name*, a *type*, and a *length*.

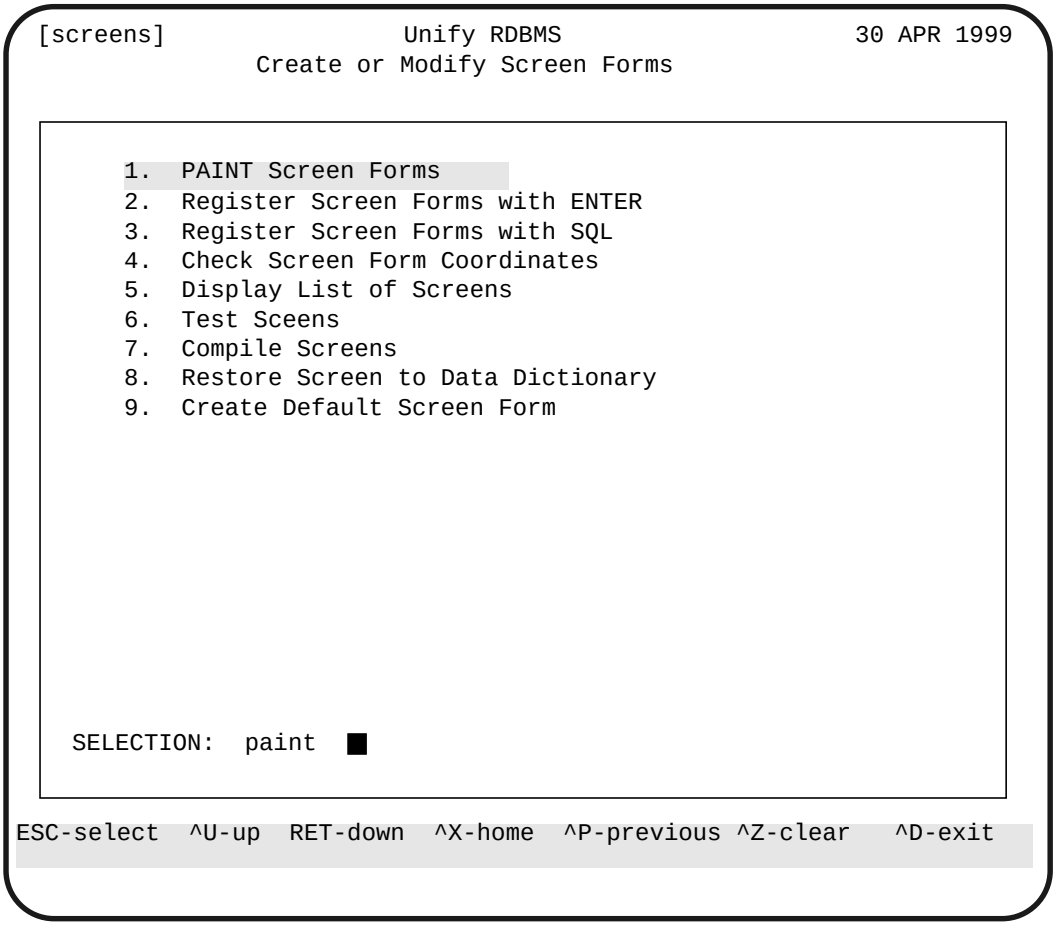
The *database field name* refers to a database field to input or display on the screen form. The same database field can appear more than once with different screen field names. This is an optional characteristic of a screen field.

The *type* and *length* refer to the data type and display length of the screen field. If you do not enter a database field name, you can enter a type and length for the screen field. However, if you enter a database field name, the type and length of the screen field are the same as the type and length for the database field. These are optional characteristics of a screen field.

To understand the differences between screen fields and database fields, work with your own screen form. The easiest way to start is to use Create Default Screen Form, which designs basic forms for you automatically. When you want to change a default form or design the form yourself, you can use Paint Screen Forms or Check. Screen Form Coordinates (see chapter 14).

Paint Screen Forms lets you move the cursor on the screen to enter prompts and screen fields where you want. You can also add special display characteristics, such as reverse video and underline. The commands PAINT uses are similar to those of full-screen text editors. You can even customize the command keys.

To use the screen form development tools, select "Create or Modify Screen Forms" (screens) from the Main Menu. The following menu and options display.

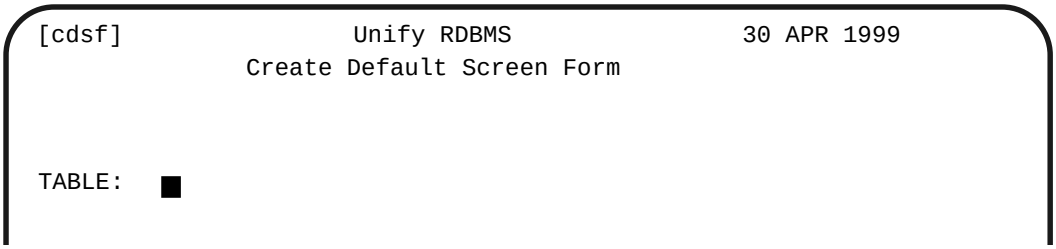


Options 1 and 9 are described in sections 13.2 and 13.1. For descriptions of options 4 through 8, see sections 14.1 through 14.4. For a description of option 2, see section 15.1. And for option 3, see section 16.4.

13.1 Create default screen form(cdsf)

Using this option, a screen form that refers to all the fields in a particular table can be created, compiled, and registered with ENTER.

Start the program by selecting "Create Default Screen Form" (cdsf) from the Create or Modify Screen Forms menu. The following screen form displays:



The TABLE: prompt is where you enter the name of an existing table. This generates a default screen form for that table. After the default screen form is created, processed, and registered with ENTER, the screen form is erased. You then can enter another table name. CTRL U returns you to the Menu Handler, while RETURN has no effect.

If all the fields for the table cannot fit on the default screen form, the following message displays:

There is not enough room on the screen for all of the fields ->->

If you enter a Y (yes) at the Continue? prompt, the default screen form is generated with the fields that fit. If you enter a N (no), the screen form is not generated and you return to the TABLE: prompt.

The following is the last message that displays before you enter the name of another table:

> The screen can now be used for data entry ->->

To use the screen form, enter its name at the selection prompt on any menu.

Because the screen form has been registered with ENTER, it can be added to any menu's list of selections. You can do this using the Add, modify or delete menus option found on the Unify DataServer/ELS Main menu (section 12.2).

You can modify or delete a default screen form with either Paint Screen Forms (section 13.2) or Check Screen Form Coordinates (section 14.1). You can also change the heading by using Register Screen Form with ENTER (section 15.1).

NOTE: When a screen form is deleted, its registration with ENTER is canceled. Any references on menus are removed as well.

In general, fields display on the finished screen form from top to bottom, as they appear in the database design. The fields line up, one underneath the other, starting after the longest prompt. The screen form's heading is the target table's long or regular name followed by the word "Maintenance." Any underscore characters are changed to spaces, and appropriate letters are capitalized.

The prompts align in one, two, or three columns. A field's prompt is its access name, which is either the field's long name or the field's regular or implicit name. In either case, all underscore characters are replaced with spaces.

An implicit name consists of a field's regular name, followed by the regular name of the field it references. The referenced field can also reference another field, thereby adding to the implicit name.

The component fields of a COMB field have separate prompts. If a field references a COMB field, a separate prompt is created for each referenced component field.

For example, suppose you want to create a default screen form for the customer table in the *Unify DataServer/ELS Developer's Tutorial Manual*. Figure 13.1 shows the customer table design.

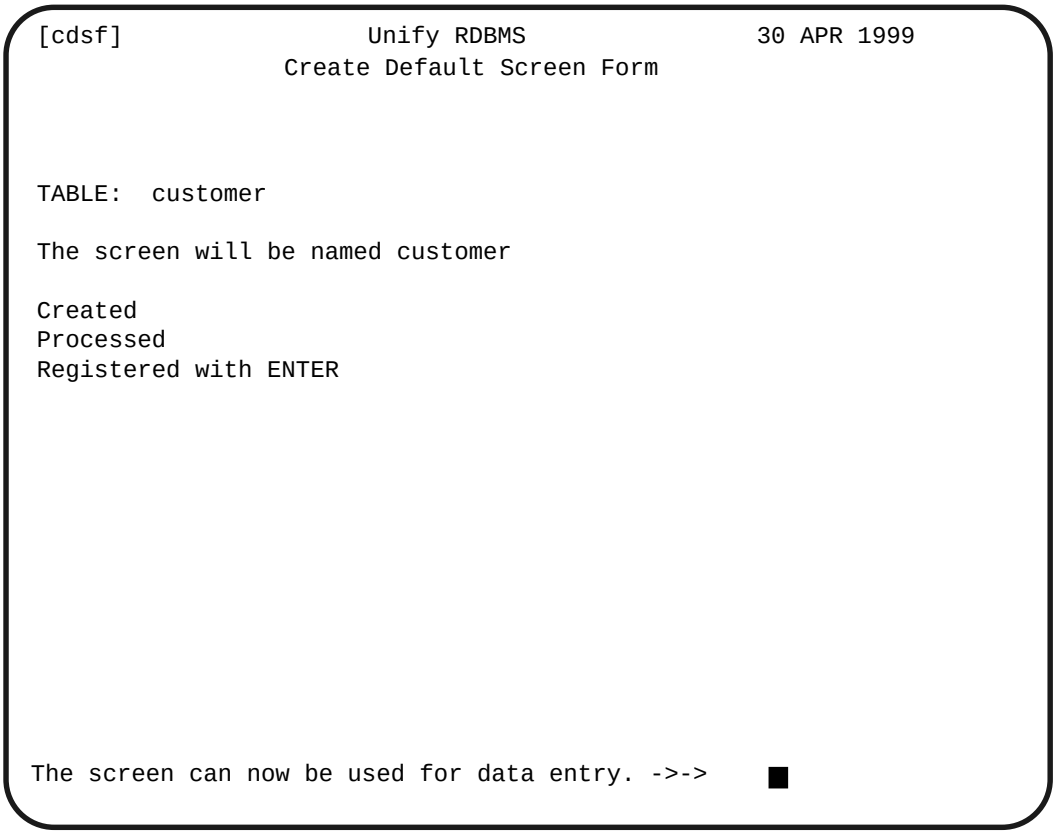
Table/field	Ref	Type	Len	Long name
customer	20			customer
*cnum		NUMERIC	5	Customer_Number
cname		STRING	30	Name
caddr		STRING	30	Address

Figure 13.1 customer table design

Table/field	Ref	Type	Len	Long name
ccity		STRING	20	City
cstate		STRING	2	State
czip		STRING	5	Zip_Code
cphone		STRING	14	Phone_Number

Figure 13.1 customer table design

Select Create default screen form (cdsf) from the Create or modify screen forms menu, and enter the table name customer at the TABLE: prompt. You see processing messages as the screen form is created, processed, and registered with ENTER. When the process is complete, the screen form displays as follows:



Press CTRL U to return to the Create or Modify Screen Forms menu, and enter customer at the selection prompt. You see the following screen form, ready for data entry:

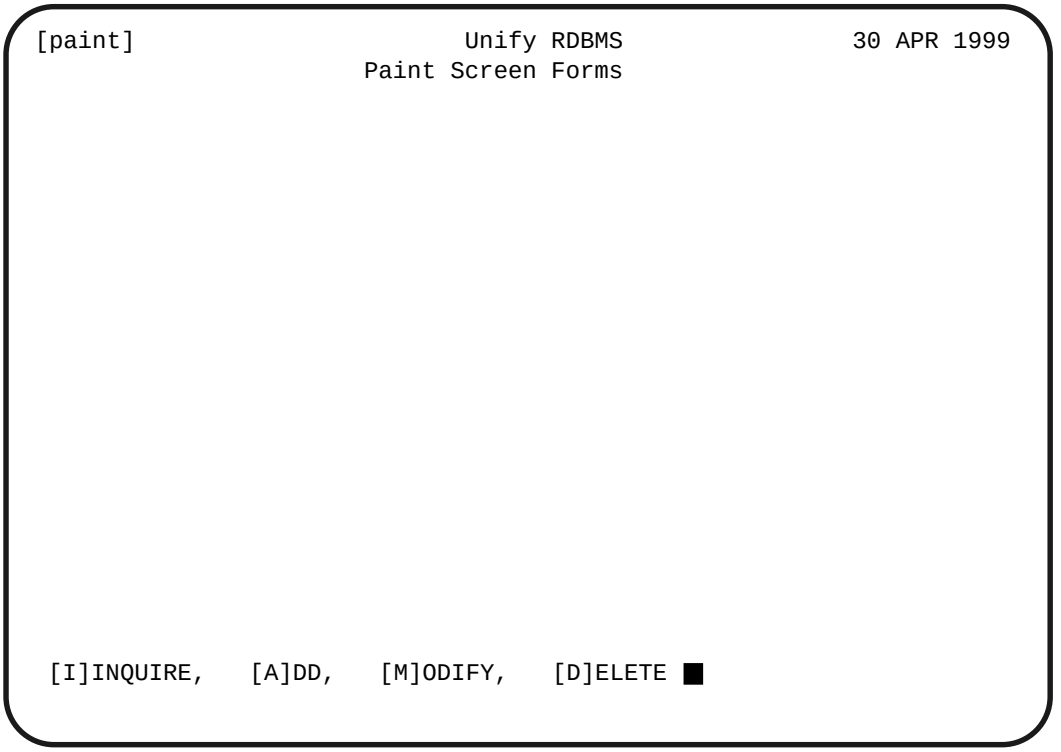
```
[customer]                  Unify RDBMS                  30 APR 1999
                        Customer Maintenance



 Customer Number:
 Name           :
 Address        :
 City           :
 State          :
 Zip Code       :
 Phone Number   :









 [I]NQUIRE,   [A]DD,    [M]ODIFY,    [D]ELETE
```

13.2 Paint screen forms (paint)

The Paint screen forms option (paint) is designed for creating, modifying, and viewing screen forms. PAINT stores and accesses the screen forms using the Unify DataServer/ELS data dictionary.

After you have specified a screen form name, you can move the cursor and add, modify, and delete prompts and screen fields. The standard commands PAINT uses are described in this chapter. You can change these if you wish, using the information described in Customizing PAINT Commands.

When you select the Paint screen forms option, the following screen displays:



You can inquire about, add, modify, and delete screen forms, prompts, and screen fields. The blank area from lines 3 to 22 is the *editing area*. This is where you can

design your own screen form. The *operation prompt* displays at the bottom of the screen:

[I]NQUIRE, [A]DD, [M]ODIFY, [D]ELETE:

Selecting any operation mode displays the current mode in the top left corner of the screen, and replaces the mode prompt with the prompt, SCREEN:. This is where you enter the name of the screen form you want to access. Pressing CTRL U at the SCREEN: prompt erases the screen and redisplays the mode prompt.

Pressing CTRL U at the operation prompt returns you to the Menu Handler.

NOTE: If you use the PAINT exit command key, changes you have made are not saved. The unicap PAINT command keys are described near the end of this chapter.

The operation modes are described as follows:

i Inquire mode. This lets you see the design of an existing screen form, but not modify it. When you finish viewing the screen, press RETURN to erase the screen and redisplay the SCREEN: prompt.

a Add mode. This lets you add new screen forms to the data dictionary, and add, modify, and delete prompts and screen fields. When you select a, a blank screen form displays and the cursor waits at the third line for you to enter a PAINT command.

m Modify mode. This lets you add, modify, and delete prompts and screen fields for existing screen forms. When you select m, the specified screen form displays and the cursor waits at the third line for you to enter a PAINT command.

d Delete mode. This lets you delete an existing screen form and all its prompts and screen fields from the data dictionary. When you select d, the specified screen form displays, as well as the prompt. Delete?. You must confirm the delete request. Respond with a y to delete the screen form. Respond with an nor a CTRL U to cancel the delete request and retain the screen form.

Deleting a screen form also removes all references to the screen form on menus, in Register Screen Form with ENTER, and in Register Screen Form with SQL.

Other prompts that display on the top line of the PAINT screen are:

F: The current screen field's name.

L: The cursor's current line position.

C: The cursor's current column position.

When you are finished editing the current screen form, press q (the *quit* command) to exit. The following prompt displays:

[S]ave, [D]on't save, [R]esume editing

If you enter s, the changes you have made are saved in the data dictionary, and the screen form is processed to produce updated .q and .h files.

If you enter d, your changes are not saved. The screen form remains the same.

If you enter r, or CTRL U, you can resume editing the screen form.

PAINT uses its own algorithms to divide the screen form into data dictionary elements. Consequently, there may be slight differences between the .q and .h files produced by Check screen form coordinates and Compile screens (sections 14.1 and 14.4). If you are using custom programs to drive your screen forms, use either PAINT or Check screen form coordinates, but not both, to create and modify screen forms.

Screen editing commands

PAINT screen editing commands can be divided into four different groups: *cursor motion, prompt editing, screen field editing,* and *miscellaneous*.

Although screen editing commands have default settings, they can be associated with different keystrokes. Customizing PAINT commands, later in this chapter, describes how to associate screen editing commands with specific keys.

The tables in Figures 13.2a through 13.5 summarize the standard screen editing commands:

Cursor motion commands

When you want to move the cursor...	Then use...
left	(left_arrow) back space h CTRL H
right	(right_arrow) space l
up	(up_arrow) k
down	(down_arrow) J CTRL J
to the next line	RETURN
to the next word	w
to the previous word	b

Figure 13.2a Cursor motion command summary

Cursor motion commands

When you want to move the cursor...	Then use...
to home position (line 2, col 0)	H
to the bottom of screen (line 22,col 0)	L
to cursor coordinates x, y (RETURN must be entered after the x and y coordinates)	g x RETURN y RETURN
to col 0 of the current line	0
to the first non-blank character on the current line	^
to the last non-blank character on the current line	$

Figure 13.2b Cursor motion command summary

Prompt editing commands

When you want to...	Then use...
enter append mode	a
enter insert mode	i
enter replace mode	R
exit append, insert, or replace modes	ESC
delete the current character	x
open a blank line above the current line	O
open a blank line below the current line	o
delete the current line	d
transfer the current line	t
toggle normal video on/off	[
toggle reverse video on/off	[
toggle underline video on/off	]
toggle reverse underline video on/off	+

Figure 13.3 Prompt editing command summary

Screen field editing commands

When you want to...	Then use...
add a new field	A
modify a field	M
delete a field	D
transfer a field	T

Figure 13.4 Screen field editing command summary

Miscellaneous commands

When you want to...	Then use...
display PAINT help	?
display the name of the current screen form	s
toggle the cursor position display on/off	c
redraw the screen	CTRL R

Figure 13.5 Miscellaneous paint command summary

Miscellaneous commands

When you want to...	Then use...
quit PAINT mode, returning to [Save], [D]on't save... prompt	q
exit to the Menu Handler, without saving	CTRL X

Figure 13.5 Miscellaneous paint command summary

The command keys can be changed by modifying your unicap file. See Customizing PAINT commands, later in this chapter.

You usually begin an editing session with cursor motion commands to position the cursor, and then use edit commands to make changes.

You should end the session by saving your changes. Because saving changes involves extensive database updating, it is a good idea to wait until you are finished editing before saving. This is somewhat different from using a text editor, where you save updates more often. The following are details on the groups of commands:

CURSOR MOTION

Cursor motion commands let you move the cursor in the editing area of the screen. In some cases, more than one key has the same effect. For example, you can move the cursor right one column by pressing the space bar, the right_arrow key (->), or l.

You can move the cursor anywhere in the editing area. Cursor motion is not restricted by blank areas, screen field prompts, or screen fields.

When the cursor is positioned on a screen field, the screen field name displays in the top right corner of the screen, directly below the screen form name.

The cursor motion commands are described as follows:

cursor left — Moves the cursor one column to the left. You can move more than one column at a time by entering a number before the command. The number is called the command repeat count. If the cursor is in column 0, it stays there.

cursor right	Moves the cursor one column to the right. You can move more than one column at a time by entering a number before the command. If the cursor is in the last column, it stays there.
cursor up	Moves the cursor up one line in the same column. You can move more than one line at a time by entering a number before the command. If the cursor is on line 2, it stays there.
cursor down	Moves the cursor down one line in the same column. You can move more than one line at a time by entering a number before the command. If the cursor is on line 22, it stays there.
next line	Moves the cursor to the first blank column of the next line.
next word	Moves the cursor to the first character of the next word. A word is defined as any sequence of characters with spaces around it.
previous word	Moves the cursor to the first character of the previous word. Alternatively, you can use the back word command.
home cursor	Moves the cursor to the top left corner of the edit area (line 2, column 0).
last line	Moves the cursor to the bottom left corner of the edit area (line 22, column 0).
go to	Moves the cursor to a specific location within the editing area. When you press the go to command key, the cursor moves to the line coordinate indicator (L:) at the top right corner of the screen. Enter a line number from 2-22 if you are using an 80-column terminal, or from 2-79 if you are using a 132-column terminal. Press RETURN to move the cursor to the column coordinate indicator (C:). Enter a column number from 0-79 if you are using an 80-column terminal, or from 0-131 if you are using a 132-column terminal. Press RETURN, and the cursor moves to the indicated position. Use CTRL U cancel the command. If you want to turn the cursor coordinate indicator on or off, press c.

column 0 Moves the cursor to column 0 of the current line.

beginning of line
Moves the cursor to the first nonblank character on the current line.

end of line Moves the cursor to the last nonblank character on the current line.

PROMPT EDITING

Prompt editing commands let you add, modify, and delete screen field prompts on screen forms. Screen field prompts are fixed strings of text that make up the basic design of a screen form. You can display prompts in normal, reverse, underline, and reverse underline video, if your terminal supports these video attributes.

To add text to a screen form, you must switch from command mode to text input mode. Once in input mode, the print characters you type display on the screen and existing characters shift to the right. Depending on whether you want to add text before or after the current cursor position, you can enter input mode two ways:

- by inserting
- by appending,

In addition to input mode, you can use replace, transfer, and video modes to edit screen field prompts. Replace mode lets you replace existing characters with characters you type. Transfer mode lets you move complete lines or screen fields to new locations on the screen form. Video mode lets you reset the video attributes of screen text or prompts.

To remind you which mode you are using, PAINT displays the current edit mode in the top left corner of the screen. You can cancel any mode by pressing the escape command key.

PAINT treats each line separately. You may notice this when insert line inserts a new line without switching to input mode, and when input mode lets you enter new text on the current line but does not let you move to a new line.

The prompt editing commands are as follows:

append input mode
: Switches from command mode to input mode, so you can enter new text after the current cursor position.

  As you enter characters, existing screen fields and prompts shift right. As the line shifts, prompt characters in the right-most column are lost. However, if a screen field is in the right-most column, you cannot enter new characters that would push the field off-screen.

insert input mode
: Switches from command mode to input mode, so you can enter new text before the current cursor position. Otherwise, this command works the same as the append input mode command.

replace mode
: Switches from command mode to replace mode, so you can enter characters that print over existing characters on the screen form.

  Existing characters do not shift right as in input mode, but are replaced. You cannot use replace mode to type over screen fields. But you can use the non-printing cursor motion command keys, such as left_arrow to move the cursor on the screen.

escape
: Cancels input, replace, transfer, and video modes and returns to command mode. If you are using command mode, escape resets the command repeat count to 0.

delete character
: Removes the character under the cursor. Characters to the right of the deleted character shift left. If the cursor is on a screen field, this command does not work.

  You can delete more than one character at a time by specifying a command repeat count.

open above
: Opens a blank line directly above the current line on the screen and shifts lower lines down. PAINT remains in command mode.

  If text is on the bottom line of the screen, the text is lost when the line above shifts down and replaces it. If a screen field is located on the

bottom line, a line cannot be inserted. The screen field must be deleted from the bottom line before a line can be inserted.

open below — Opens a blank line directly below the current line. Otherwise, this command works exactly like the open above command.

delete line — Deletes the current line and shifts the lines below it up one line. If the current line contains a screen field, the screen field must be deleted with the delete screen field command.

transfer line — Transfers the current line to a new position on the screen form. The line can contain both prompts and screen fields.

To move a line, press the transfer key to start the transfer command, move the cursor to a new position, and press the transfer command key again. The line is then moved from its old position and to the new position.

You can use such nonprinting cursor motion command keys as left arrow to move the cursor. The transfer command can be canceled at any time with the escape command.

VIDEO CONTROL

You can set video display attributes with the normal, reverse, underline, and reverse underline video commands:

normal video — Displays characters in normal intensity, bright characters on a dark background.

reverse video — Displays characters in reverse video, dark characters on a light background.

underline video — Underlines characters.

reverse underline video — Displays characters in reverse video and underlined.

The video control commands work as follows:

1. Move the cursor to one end of the area where you want to change the video attribute.
2. Enter the appropriate video command key, to cause a mark character to display.
3. Move the cursor to the other end of the area, either to the right or the left of the mark character. You can use any of the cursor motion commands that don't move the cursor to another line.
4. Type the same video command key again. The characters from the mark character to the cursor display in the selected video attribute.

   You can cancel these commands at any time with the escape command.

NOTE: You must have the appropriate video entries defined in the termcap file (see section 6.1). The default mark character is set to the tilde (~).

Some terminal types, such as the TeleVideo 912, 920, or similar types, use extra character positions on screen to mark the start and end of reverse and underline video attributes. These attribute characters look like ordinary spaces on the screen, but must be treated different from normal spaces.

PAINT doesn't allow attribute characters to be shifted off the screen, deleted, or replaced. This would change the video display attributes of the area, because it would no longer be marked.

You can, however, delete attribute characters by setting the video display attribute to normal. Also, these terminals can't display two reverse or underline areas separated by a single normal video space. You need to enter at least two spaces between areas with the same video attributes. Therefore, if you are using one of these types of terminals, you must leave enough space between areas displayed in reverse or underline video.

SCREEN FIELD EDITING

Screen field editing commands let you add, modify, and delete fields on a screen form. Screen fields are the windows to the database that let you view the data values of the current record.

Screen fields have a screen field name, a database field name, a type, a length, and display coordinates. When you specify the screen field name and the database field name, PAINT determines the type, length, and display coordinates.

To add or modify screen fields, switch from command mode to screen field edit mode. In this mode, the bottom two lines of the screen form are replaced with the field edit prompts:

```
SCREEN FIELD:                TYPE:                LENGTH:
DATA BASE FIELD:
```

where SCREEN FIELD: and DATABASE FIELD: are where you enter the screen field and database field names. The screen form redisplays when you finish the entry.

Screen fields must not overlap each other or prompts. You must not add or move screen fields so they overlap. Examples of the display formats for different types of screen fields are as follows:

NUMERIC	NNNNNN
AMOUNT	AAAA.AA
FLOAT	FFFFFF.FFF
DATE	MM/DD/YY
LDATE	DD/AAA/YYY
TIME	HH:MM
STRING	SSSSSS
TEXT	XXXXXXXXXXXXXXXXXXX
BINARY	B

The display length of the database field determines the number of characters displayed for the screen field. If the underline video attribute is defined in the termcap file, screen fields are underlined on PAINT screen forms. However, if a terminal requires the ug#1 entry in termcap screen fields are not underlined.

The screen field editing commands are as follows:

delete screen field

Deletes the current screen field. The current screen field name displays in the top right corner of the screen beside F:. "Current screen field" means the cursor is positioned on that field.

transfer screen field

Transfers the current screen field to a new position. After you enter the command, move the cursor to an appropriate position and press the transfer field command key again. You can use any cursor motion commands to move the cursor.

If the field does not fit in the new position, PAINT lets you position the cursor again.

The transfer command can be canceled at any time with the escape command.

add screen field

Switches from command mode to screen field edit mode.

To add a screen field, position the cursor on a blank area where there is enough space for the field. When you enter this command, the last two lines on the screen are erased and the field edit prompts display.

modify screen field

Switches from command mode to screen field edit mode.

To modify a screen field, position the cursor on an existing screen field. When you enter this command, the last two lines on the screen are erased and the field edit prompts display.

FIELD EDIT PROMPTS

The field edit prompts that display at the bottom of the screen when you enter the add screen field or modify screen are used as follows:

SCREEN FIELD

A unique 8-character screen field name. The name is used to refer to the screen field in programs you may write, so it must be different from

database field names. However, different screen forms can use the same screen field names. You can have as many screen fields as available memory allows.

DATABASE FIELD
: The name of the database field to display for this screen field. PAINT looks up the name in the data dictionary to validate it.

  A COMB type field cannot be used directly; you must enter each component of the COMB field separately.

  If you leave the database field blank, you must complete TYPE and LENGTH prompts.

  You can clear the database field name by entering a space.

TYPE
: The type of the database field (NUMERIC, FLOAT, AMOUNT, DATE, LDATE, TIME, STRING, TEXT, or BINARY).

  If you specify a database field name, PAINT enters the type from the data dictionary, and you cannot modify it.

LENGTH
: The display length of the database field.

  If you specify a database field name, PAINT gets the length from the data dictionary, and you cannot modify it.

The RETURN key moves the cursor forward over a prompt, and CTRL U moves the cursor backward.

If you press CTRL U at the DATABASE FIELD: prompt in modify mode and a database field has been specified, you return to command mode. The last two lines of the screen redisplay, and the screen is updated to reflect changes.

MISCELLANEOUS COMMANDS

Miscellaneous commands perform functions not directly related to creating or editing screen forms. These commands are described as follows:

help
: Displays a summary of the available commands. Two help screens display the default unicap settings for prompt editing, screen field editing, miscellaneous, and cursor motion commands.

  To view the second help screen, press RETURN. To return to the screen form you are editing, press RETURN at the second help screen.

  If you do not want to view the second help screen, enter an n at the prompt.

display screen name
: Displays the name of the current screen form at the bottom of the screen.

toggle cursor
: Turns the cursor coordinate indicators ON and OFF.

position display
: When ON, the column coordinate displays after the C: prompt and the line coordinate after the L: prompt.

  When OFF, neither the C: and L: prompts nor the column and line coordinates display.

redraw screen
: Clears the edit area and displays the screen as PAINT currently defines it.

  If words are writing over each other, and lines are not where they should be on the screen, this command should adjust the screen display.

quit
: In command mode, terminates current screen form edit. When you enter this command, the following prompt displays:

  [S]ave, [D]on't save, [R]esume editing

  Enter s to save your changes in the data dictionary and process the screen form to produce updated .q and .h files.

Enter d to drop the changes and keep the screen form the same. Enter r or CTRL U to resume editing the screen form.

exit — Returns immediately to the menu and drops all changes made since the last save.

Customizing PAINT commands

In PAINT, commands are entered with keystrokes (command keys) described in the PAINT section of the unicap file. The PAINT section of the unicap file associates commands with specific keystrokes. When you enter keystrokes associated with particular commands, PAINT performs those commands. You can easily change command definitions to use different command keystrokes by editing the unicap file. See section 6.2 for a general description of unicap.

The internal command name and action of each PAINT command in the unicap file are shown in Figure 13.6.

Name	Action
addfld	add screen field
app	start append input mode
bot	move to last line
co10	move to column 0
cpdisp	toggle cursor position display ON/OFF
delch	delete current character
delfld	delete current screen field
delln	delete current line
down	move cursor down
endln	move to end of line
escape	escape (end current mode)
exit	exit to the Menu Handler
goto	go to screen coordinates x,y
help	print PAINT help
home	move cursor to home position
ins	start insert input mode

Figure 13.6 PAINT command names and actions

Name	Action
left	move cursor left
modfld	modify current screen field
nrmlon	start normal video
nxtln	move to next line
nxtw	move to next word
oplna	open line above
oplnb	open line below
prvw	move to previous word
quit	quit paint mode
redraw	redraw screen
rep	start replace mode
right	move cursor right
rvon	start reverse video
rvulon	start reverse underline video
screen	display the system name of the current screen
stln	move to beginning of line
trnfld	transfer current screen field
trnln	transfer current line
ulon	start underline video
up	move cursor up

Figure 13.6 PAINT command names and actions

The standard unicap PAINT section is shown in Figure 13.7

SECTION=paint	paint info
ignoremiss = FALSE	report missing command definitions
mrkchar = ^	
co10 = 0	column 0
help = ?	help
delch = x | <del_char>	delete character
oplnb = o	open line below
oplna =0	open line above

Figure 13.7 Standard unicap PAINT section

SECTION=paint	paint info
ins = i	insert mode
app = a	append mode
rep = R	exchange/replace mode
delln = d | <del_line>	delete line
delfld = D	delete field
trnln = T	transfer field
trnln = t	transfer line
modfld = M	modify field
addfld = A	add field
redraw = ^r	redraw screen
cpdisp = c	change cursor position display
escape = \E	escape
exit = ^x	exit
quit = q	quit
goto = g	go to x,y
left = h |\b|^h| <left_arrow>	cursor left
right = 1 |\040 | <right_arrow>	cursor right
up = k | <up_arrow>	cursor up
down = j | <down_arrow>	cursor down
stln = \^	start of line
endln = $	end of line
nxtw = w	next word
prvw = b	previous word
home = H	cursor home
bot = L	cursor bottom
nextln = \n|\r	next line
rvon = [	reverse video on
ulon =]	underling on
rvulon = +	reverse video + underlining on
nrmlon = n	normal video on
screen = s	screen name

Figure 13.7 Standard unicap PAINT section

All PAINT command names can be associated with command keys of you choice. For example, the following line in the unicap file defines the command key to enter insert input mode:

```
ins = i
```

where ins is the command name and i is the command key. If you want insert input mode to start when you press E instead, change the i to an e. The restrictions that apply to assigning command keys to commands are as follows:

1. Digits cannot be assigned as command keys. This is because digits indicate the command repeat count.

   The exception is 0, which can only be assigned to the column 0 command.

2. ignoremiss = TRUE

   Without this line, PAINT prints a message for each command not assigned to a key. With this line, PAINT suppresses the messages.

   Commands not assigned to keys cannot be performed. The exceptions are the escape and quit commands, which are by default ESC and q, respectively.

3. mrkchar = string

   where string is one print character.

   This line lets you change the mark character that indicates the initial cursor position for such commands as add field, transfer field, transfer line, and reverse video. The default marker character is the tilde (~).

Chapter 14: Screen form maintenance utilities

This chapter describes several screen form maintenance utilities: Check screen form coordinates, Display list of screens, Test screens, Compile screens, and Restore screen to Data Dictionary. These screen form maintenance utilities are all options on the Create or modify screen forms menu.

Check screen form coordinates is an alternative to PAINT. Both programs let you create, modify, or view screen forms. However, when you use Check screen form coordinates, you must specify the x-y coordinates of prompts and screen fields.

Display list of screens lets you list existing screen forms.

Test screens lets you make sure screen field prompts and screen fields are where you want them to display on the screen form.

Compile screens and restore screen to Data Dictionary let you update the data dictionary to reflect any changes you have made to screen forms.

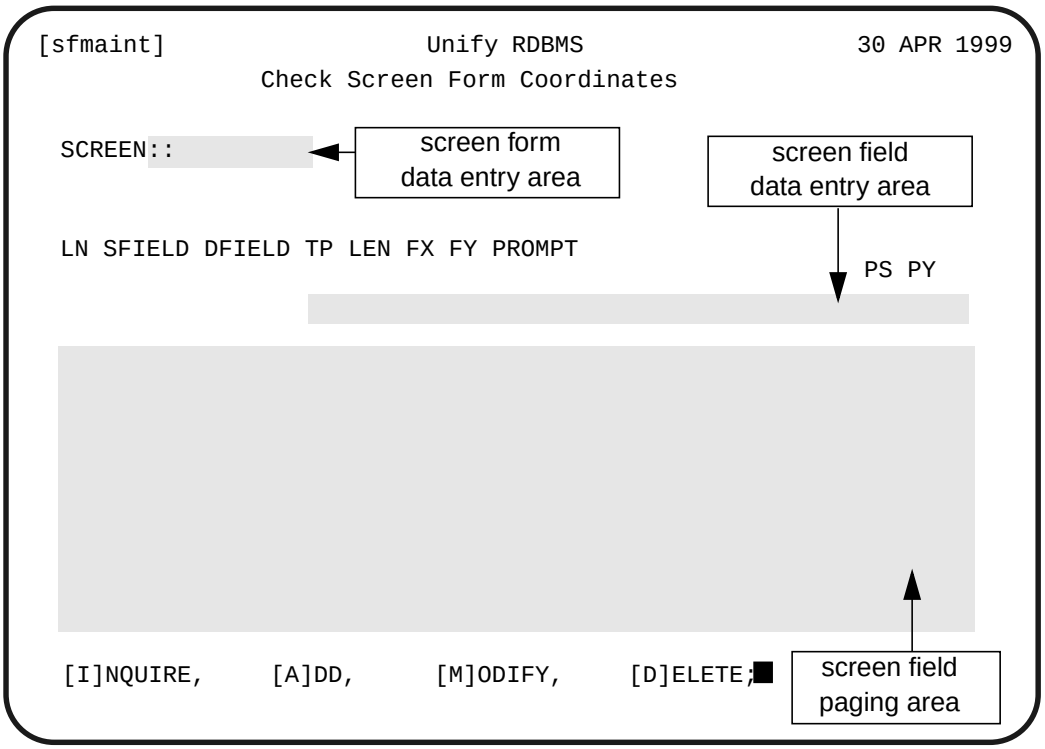
14.1 Check screen form coordinates (sfmaint)

Check screen form coordinates is an alternative method to PAINT that lets you inquire, add, modify, and delete screen forms and screen fields.

Check screen form coordinates displays paged information about a specific screen form rather than the actual screen form. With this tool, you can move to the next or previous page, select a particular screen field to modify or delete, add new screen fields, or delete screens.

Start this option by selecting Check screen form coordinates (sfmaint) from the Create or modify screen forms menu.

The following screen form displays:



The Check screen form coordinates screen form has three parts:

1. A screen form data entry area. This is where you enter the name of the screen form you want to use.
2. A screen field data entry area. This is where you enter the information for the screen fields you want to add, modify, or delete.
3. A screen field paging area where screen field information is listed. You can display the list of screen fields page by page.

The prompts and column headings on the screen are described as follows:

[I]NQUIRE, [A]DD, [M]ODIFY, [D]ELETE:

The operation prompt. This prompt lets you choose an operational mode for the screen form data entry area of the screen. The meanings of the various operation modes are as follows:

i	Inquire mode. This lets you see the screen fields for an existing screen form.
a	Add mode. This lets you add new screen forms and associated screen fields.
m	Modify mode. This lets you add, modify and delete screen fields for an existing screen form.
d	Delete mode. This lets you delete an existing screen form and all its screen fields. Deleting a screen form also removes all references to the screen form in menus, in Register Screen Form with ENTER, and Register Screen Form with SQL.

SCREEN A unique 8-character screen form name. This name is used to refer to the screen form throughout the system.

[N]ext page, [P]rev page, [A]dd line, or number:

The paging prompt. This displays in modify mode, after you specify a screen form at the SCREEN: prompt. The paging prompt lets you

choose the operational mode for the screen field paging area. The meaning of each selection is as follows:

n	Displays the next page of 12 screen fields
p	Displays the previous page of 12 screen fields
a	Lets you add screen fields. The cursor moves to the *screen field data entry area* so you can add a field.
1-99	Displays the page containing the indicated screen field, and positions the cursor at that screen field to let you modify or delete it.

LN — A line number assigned by the system. Use this number to refer to a line you want to modify.

column between LN and SFIELD

Because of space restrictions, there is no name for this column. But it is the same column as the CMD column on the table entry screen. This is where you enter operation commands for the current screen field. The cursor position marks the current screen field.

In the paging area, you can move the cursor up and down by pressing CTRL U and RETURN in this column. Valid commands in this column are as follows:

m	Modify DFIELD, TP, LEN, FX, FY, PROMPT, PX, or PY for the current screen field
d	Delete the current screen field
q	Redisplay the paging prompt at the bottom of the page

SFIELD — A unique 8-character screen field name. This name is used to refer to the screen field in custom programs, so it should be different from database field names. However, different screen forms can use the

same screen field names. You can have as many screen fields as the buffer and screen can support.

DFIELD — The name of the database field to display for this screen field. If it is left blank, you must fill in the TP and LEN prompts. The program validates the name in the data dictionary.

COMB type fields cannot be used directly; you must enter each component of the field separately.

TP — The type of the database field. This is filled in from the data dictionary if a database field name has been specified.

LEN — The display length of the database field. This is filled in from the data dictionary if a database field name has been specified.

FX — The x coordinate (column) of the database field on the screen form. It is a number between 0 and 79 for 80-column screens, and 0-131 on 132-column screens.

FY — The y coordinate (row) of the database field on the screen form. It is a number between 0 and 23.

PROMPT — The text to be displayed on the screen form to indicate what data is being entered or displayed.

You can enter literal text. Or, if your terminal supports special video modes, you can enter escape characters that indicate the prompt should be displayed in a special video mode. Escape characters consist of the tilde (~) followed by a single character specifying the video mode. These are the same escape characters described for the termcap file in section 6.1.

To display a prompt in a special video mode, enter the correct escape characters before and after the prompt. The table in Figure 14.1 summarizes the escape characters that specify special video modes.

When you want to...	Then enter...
start reverse video	~r
end reverse video	~s
start underline	~u
end underline	~v
start reverse underline video	~v
end revere underline video	~x

Figure 14.1 Special video mode escape characters

If you start a special video mode for a prompt, you must be sure to use the appropriate escape character to end that video mode. Otherwise, your screen form will not display correctly.

If you created the original screen form with Create default screen form, the database field's access name is used as the field prompt. For most fields, the access name is the long name of the field.

However, a field that references a COMB type field in another table must have a separate access name for each implied field. The access name of each implied field is a compound of the short name of the field it is part of and the short name of the component field it references in the other table.

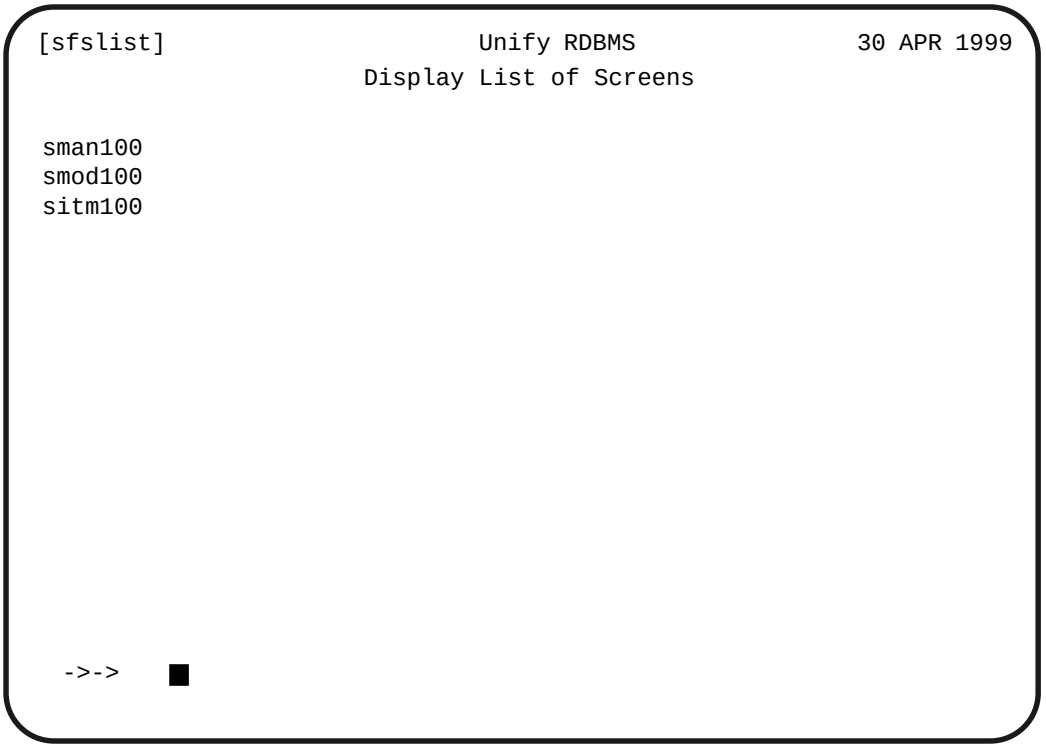
PX — The x coordinate (column) of the prompt on the screen form. It is a number between 0 and 79 for 80-column screens, and 0-131 for 132-column screens

PY — The y coordinate (row) of the prompt on the screen form. It is a number between 0 and 23.

14.2 Display list of screens (sfslist)

This program generates a report of all the screen forms in the data dictionary. It has no selection options or prompts that you must answer before the program runs.

Start the program by selecting Display list of screens (sfslist) from the Create or modify screen forms menu. The following is an example of what you should see:

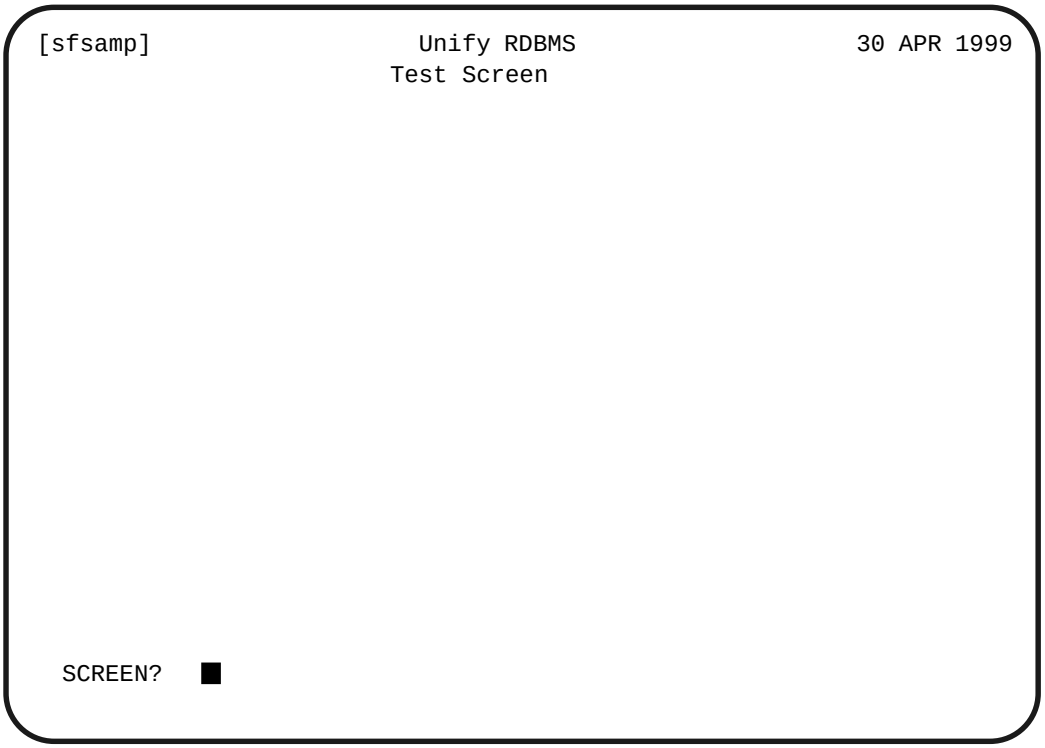


The program displays the screen form names in columns. Press RETURN to return to the Create or modify screen forms menu.

14.3 Test screen (sfsamp)

This program lets you test a screen form without writing a program to display the form. The position of screen field prompts and screen fields can be easily checked in this way.

Start the program by selecting Test Screens (sfsamp) from the Create or modify screen forms menu. The following screen form displays:



Enter the name of the screen form you want tested at the SCREEN: prompt. The screen form displays, with an x indicating the position of each screen field.

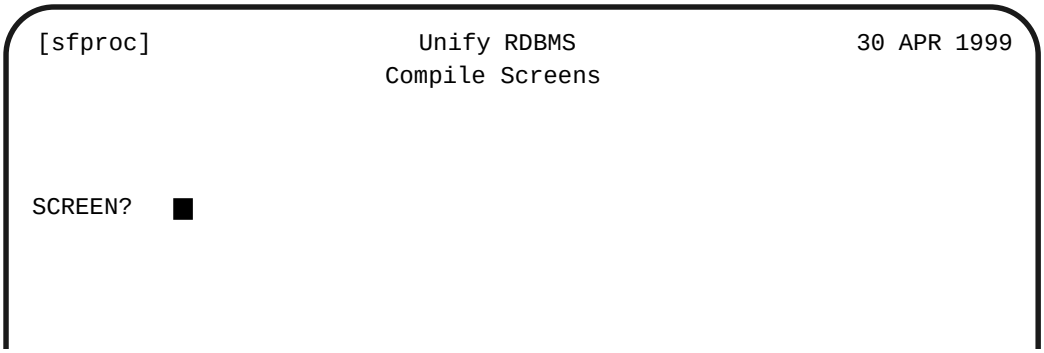
Either enter the name of another screen form to test at the SCREEN: prompt, or press CTRL U to return to the Create or modify screen forms menu.

14.4 Compile screens (sfproc)

This program creates two screen form files used by application programs: a .h and a .q file. The .h file, or include file, is used by C programs to reference screen field names. The .q file contains a binary version of the screen form that can be read quickly at run time. For more information about these files and how they are used, see the *Unify DataServer/ELS HLI Programmer's Manua*l.

You should run this program after you modify a screen form with Check screen form coordinates. However, PAINT automatically compiles the screens.

Start this program by selecting Compile screens (sfproc) from the Create or modify screen forms menu. The following screen form displays:



At the SCREEN: prompt, either enter a screen form name to compile a specific screen, or enter all or ALL to compile all screen forms in the data dictionary.

Each screen form is then compiled, and the .h and .q files are created. After the process, the screen clears, so you can enter another screen form name. Press CTRL U to return to the Create or modify screen forms menu.

14.5 Restore screen to Data Dictionary (sfrestr)

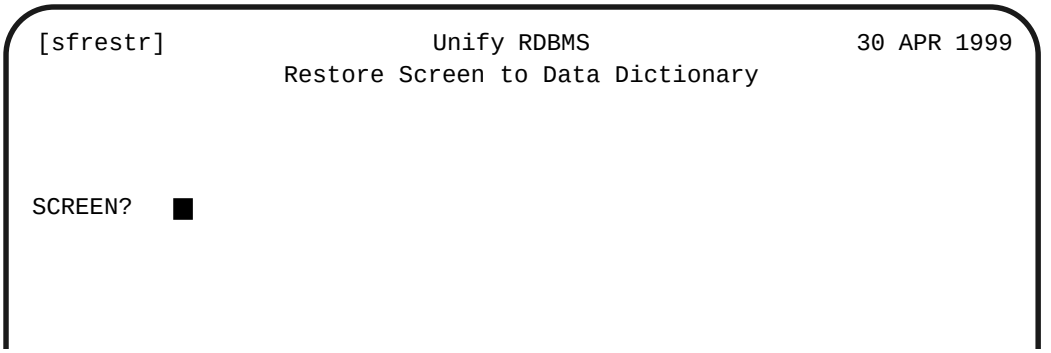
This program lets you recover screen forms inadvertently deleted from the data dictionary, or to reorder the prompts on a screen.

The order of prompts must conform to the following rules:

- The primary key (if any) must be the first prompt.
- Combination fields and B-trees must be moved as a unit. You cannot reorder the fields within a COMB data type.

If you have made many changes and the prompts are out of order, first process the screen (Compile screens) and delete the screen (Check screen form coordinates). Then restore the screen using this program. The program reads the .q file in the current directory, and inserts the form back into the data dictionary. Screen field prompts are reordered sequentially.

Start the program by selecting Restore screen to Data Dictionary (sfrestr) from the Create or modify screen forms menu. The following screen form displays:.



At the SCREEN: prompt, enter a screen form name to recover the screen form from the .q file. After the screen form has been recovered, the screen clears, so you can enter another screen form name. Press CTRL U to return to the Create or modify screen forms menu.

Part V: Data entry and retrieval

Part V, Data entry and retrieval, contains chapters 15 and 16.

Chapter 15, ENTER-data entry/query by forms, describes how you can use ENTER screens to query the database, add or modify data, and print reports.

Chapter 16, SQL-query/dml language, describes the Unify DataServer/ELS query language, SQL, and Unify DataServer/ELS data manipulation language, DML.

Chapter 15: ENTER–data entry query by forms

This chapter explains ENTER and how it can be used. ENTER is a general-purpose program that enables you to develop data entry and inquiry screen forms without writing a custom program. It lets you add data to a database quickly and efficiently, and provides a powerful and easy-to-use query interface.

This interface enables you to find and display data using a screen form. Once the data is selected, you can update, delete, or summarize it in a report.

This chapter describes the features that make ENTER so powerful and easy-to-use:

1. You can register a screen form with ENTER, and associate the screen form with RPT report scripts and custom programs. For more information about RPT, see chapter 17. For more information about custom programs, see the *Unify DataServer/ELS HLI Programmer's Manual.*
2. You can then use the screen form for data entry and inquiry.
3. ENTER interfaces with Advanced field attributes (section 8.6). If you use Advanced field attributes, ENTER can:
    - Validate values for records as they are added or modified
    - Assign default values for fields
    - Display a custom error message for invalid values
    - Display an information message and a help file for each field
4. You can use ENTER query by forms to select database records that you want to inquire about, modify, or delete.
5. ENTER interfaces with RPT (chapter 17). When you register a screen form with ENTER, you can associate the screen form with report scripts. You can then print the reports for records selected with query by forms.

6. ENTER screens interface with the Menu Handler (chapter 3). Therefore, program level security can be applied to these screens as well.

Before you can use these features with ENTER, you should understand the rules that ENTER is guided by.

The Rules of Using ENTER

ENTER is guided by rules that determine what order fields are accepted and validated on the screen form. The rules also determine whether or not ENTER can work with the screen form.

It is entirely possible to define screen forms that ENTER cannot use. In this case, you can write a host language program to use the screen form.

Normally, an ENTER screen form works with a single target table. A target table is the table that the screen form maintains — the one inquired, added, modified, or deleted. An ENTER screen form can display related information from fields in secondary tables, but these fields are display-only and cannot be edited.

ENTER can only display information from secondary tables if there is an explicit relationship between the target and the secondary table.

The basic rules for developing screen forms you want to use with ENTER are as follows:

1. An ENTER screen can update the designated target table only. The target table is defined with Register Screen Form with ENTER. Only target table records can be added or deleted.

   Fields from secondary tables are display-only. A secondary table field can be displayed only when the previous target table field on the screen form has an explicit relationship to a field in the secondary table.

In the following example, the manufacturer name only displays on the model screen form because it is preceded by the manufacturer ID number:

```
[paint]                    Unify RDBMS                        30 APR 1999
Command mode             Paint Screen Forms              F:  L:2  c:0

■


Model Number      :  NNNNNNN
Manufacturer ID   :  NNNN                  manufacturer name
Description       :  SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
```

The manufacturer ID number is an explicit reference from the model to the manufacturer. This is indicated in the database design for model with a reference to mano in the REF column, as shown in Figure 15.1:

Table/field	REF	Type	Len	Long name
manf	10			manufacturer
*mano		NUMERIC	4	Manufacturer_ID
mname		STRING	35	Name
madd		STRING	30	Address
mcity		STRING	20	City
mstate		STRING	2	State
mzip		STRING	5	Zip_Code
model	50			model
*mokey		COMB		Model_Key
monum		NUMERIC	7	Model_Number
momano	mano	NUMERIC	4	Manufacterer_ID
mdes		STRING	30	Description

Figure 15.1 Database design with explicit relationship

2. If a combination field in the target table has an explicit relationship to a secondary table, ENTER validates all components of the combination field before storing the combination field in the database.

That is, ENTER makes sure values entered to combination field components match existing values in referenced fields.

When individual components of a combination field are on the screen form, each component is entered in sequence. Therefore, all the components of a combination field should be together on a screen form so the cursor can move from one to the next without jumping around.

3. In ADD mode, the target key field must be entered before any other fields. If the target key is a combination field, all its components must be entered and validated.

   Therefore, you should make sure the target key is the first field on the screen form.

4. Screen fields must have corresponding database fields. ENTER ignores screen fields that lack database fields.

5. ENTER uses the last three lines of the screen form to display information, prompts, and error messages. If you use lines 21, 22, or 23 to display your data, ENTER overwrites it.

   NOTE: The following rule applies only if you use Check screen form coordinates to develop the screen form:

6. You must specify both the database field coordinates, FX and FY, and the prompt coordinates, PX and PY, for all screen fields, even if you do not specify a prompt.

   ENTER uses the FX-FY combination to clear data from the screens of terminals that do not have protected fields. ENTER uses the PX-PY combination to order the fields on the screen form and determine how the cursor moves from one field to another.

   If you leave FX and FY at zero, the top line of the screen erases when ENTER clears the screen. If you leave PX and PY at zero, the cursor moves erratically from one field to another.

15.1 Register screen form with ENTER (entmnt)

Before ENTER can use a screen form, you must tell ENTER about the screen form and how it is to be used. This is known as the registration process.

The registration process associates the screen form with a target table and a title. You must specify the target table, because fields from several secondary tables can also display on the screen form.

A screen form can also be registered with SQL (chapter 16), but it may not be registered with both ENTER and SQL. To confirm whether an existing screen form is registered with ENTER or SQL, run Print list of programs (section 11.7).

When you register a screen form with ENTER, you can also associate an optional list of reports with the screen form. Each report has a formatting program, parameters, title, and a set of allowable report output destinations. These reports can then be run using the Report options screen, described later in this chapter.

Warning!

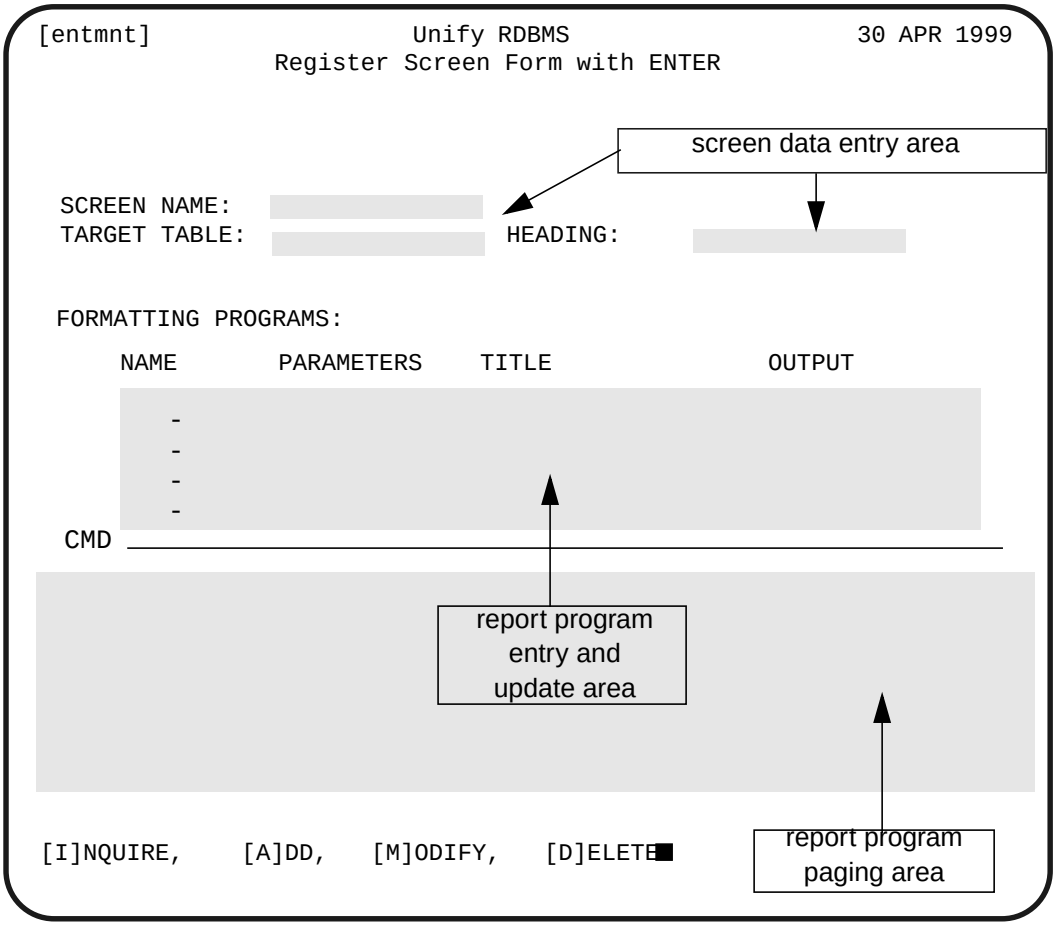
The registration process creates an entry for the screen in the data dictionary. The registered screen is an ENTER screen. Once registered, an ENTER screen can be used as an option in a menu (section 12.2).

Warning!

When you delete an ENTER screen from the data dictionary, all menu references to the screen form are removed. You can list all the menus in the system using Print Menus (section 11.2).

Warning!

Start this option by selecting "Register Screen Form with ENTER" (entmnt) from the Create or Modify Screen Forms Menu. The following screen form displays:



The prompts and column headings on this screen form are described as follows:

[I]NQUIRE, [A]DD, [M]ODIFY, [D]ELETE:

The operation prompt. The following operation modes can be entered at this prompt:

i	Inquire mode. This lets you view an ENTER screen's registration data.
a	Add mode. This lets you register screen forms with ENTER.
m	Modify mode. This lets you modify TARGET TABLE, HEADING, and FORMATTING PROGRAMS information for a screen form previously registered with ENTER,
d	Delete mode. This lets you remove the ENTER registration for a screen form.

Warning!

Deleting an ENTER registration removes all references to the screen form, such as menu listings. However, it does not affect the original screen form.

See section 13.2 or 14.1 for information on deleting screen forms from the data dictionary, and section 11.2 for information on listing menus.

SCREEN NAME

The name of the screen form to register.

TARGET TABLE

The name of the table that the screen form maintains— the one inquired, added, modified, or deleted. You must specify the target table because fields from secondary tables can also display on the screen form.

HEADING A string of up to 34 characters that displays on line 2 of the screen form when ENTER is running. The heading also displays on each menu where the ENTER screen is a selection option.

FORMATTING PROGRAMS

The area below this prompt is for updating and displaying the report-generating programs associated with this ENTER screen. The combination of the formatting program NAME, PARAMETERS, TITLE, and OUTPUT options is called a report registration.

Report names registered with an ENTER screen display on the Report options screen. For more information about the Report options screen, see chapter 15.

NAME

The first prompt in the update area below FORMATTING PROGRAMS. You can specify an 8-character name for the executable formatting program that is used to produce the report. You can associate up to eight different formatting programs with each screen registration.

The formatting program can be the listing processor (LST), the report processor (RPT), a custom program, or a blank. LST and RPT must be in upper case letters. A Unify DataServer/ELS executable name other than RPT or LST is not accepted.

If you enter a blank, no formatting program will be used. The report will list all fields on the screen form in a tabular format, 80 characters wide.

NOTE:

If the length of all the fields in a record exceeds 80 characters, extra characters wrap to the next line.

This field is edited so that only the characters before a blank remain. When you press RETURN the string is left-justified. Leading spaces and extra characters are deleted.

PARAMETERS

Four 14-character strings that can be passed as parameters to the formatting program. The formatting program must interpret the meaning of the parameters. Parameters are entered one string per row, next to each hyphen (-). The type of formatting program determines the number of parameters allowed. Figure 15.2 shows the number of parameters allowed for each type of formatting program.

When the FORMATTING PROGRAM is...	Then the maximum number of PARAMETERS is...
RPT	1(the name of an RPT script)
LST	1(the name of an LST script)
blank	0
a custom program	4

Figure 15.2 Parameters per formatting program

An RPT or LST script name entered as a parameter must be one word with no embedded or leading spaces.

If entering parameters for a custom program, the parameters are not edited. You can place more than one parameter in each PARAMETERS field by leaving a space between each parameter, provided the total length does not exceed 14 characters.

The custom program is then started as if the following command was entered at the operating system level:

```
program sel-file p1 p2 p3 p4
```

where *sel-file* is the name of the selection file created by the user's query by forms query or LST selection processor query, and *p1-p4* are the four parameter strings.

For information on the type of functions you can use with the *sel-file*, see Selection functions in the *Unify DataServer/ELS HLI Programmer's Manual*

TITLE — A descriptive label for the report. This field displays on the Report options screen (section 15.5) to identify the report by its selection number.

OUTPUT — This sets the allowed destinations for report output and sets the optional debugging mode. Output can be routed to more than one destination per run. The following four characters can be entered:

S Enables the screen output option (DEFAULT)

P Enables the primer output option

F Enables the file output option

N Disables the debug option, no-debug (DEFAULT is enable debug)

Lower case entries are converted to upper case. Duplicate entries are ignored.

Running the report in debug mode displays any error messages, the report scripts used, and runtime results. When debug mode is enabled, you can verify the report script and see syntax errors.

CMD

The area under this text prompt is called the display area. The first line of each registered report displays here in the order it appears on the ENTER Report Options Screen.

Although you add and modify individual reports in the update area, you delete reports directly from this area.

The control commands you can enter for a line under CMD are as follows:

q	Stop the current mode and leave the display area. This command can be entered on any line.
d	Delete the current report from this screen form's ENTER registration.
a	Add a report. The new report is to be associated with this screen form's ENTER registration. You can only enter an a on a blank line at the end of the display area.
#	A number from I to n, where n is the number of reports registered with this screen form. The current item is moved to the specified line number and all reports are renumbered after the move.
m	Modify fields. Lets you modify all fields associated with the current report.
NOTE:	If you change NAME to a blank, all existing parameter fields are erased. If you change NAME to a Unify DataServer/ELS-supplied formatting program, all fields except the first are erased.

15.2 Using ENTER for data entry

Data entry is the process of adding data to a database or modifying existing data. ENTER can be best understood by example, rather than by description. For an example of how to use a specific ENTER screen, see the *Unify DataServer/ELS Developer's Tutorial Manual.*

An ENTER screen has four operation modes: *inquire, add, modify,* and *delete*. One or more of the modes may be disallowed depending on your access privileges defined with Add or Modify Group or Individual Privileges.

A typical ENTER screen works in the following way:

Entering the primary key field

In add mode: The cursor moves first to the key field. You must enter the key before you can move the cursor down the screen. If the key is a COMB field that has several parts, you must enter all parts. After you enter the key, the record is added to the database and locked so other users cannot modify or delete it. You can then enter the remaining target table fields on the screen form.

In modify mode
The cursor also moves first to the key field. The difference is that you can use Query By Forms to locate records. After the records are located, the screen operates as in add mode.

NOTE If another user is modifying or deleting a record you are trying to modify, an error message displays. You cannot update the record. When the other user finishes with the record, you can try again.

Moving the cursor on the ENTER screen

After a record has been added or located, the cursor moves to the next field prompt on the screen. Press RETURN to move the cursor down the screen, and CTRL U to move it up the screen.

Note that the cursor skips fields from tables other than the target table, because they are display-only.

When you reach the last prompt, press RETURN to loop the cursor back to the first field prompt. Press CTRL U at the first field prompt to clear the screen form for the next operation. In add or modify mode, this saves the record to the database. If you press CTRL U again, the operation prompt [I]NQUIRE, [A]DD, [M]ODIFY, [D]ELETE redisplays.

At any time, you can press CTRL X to immediately return to the Menu Handler.

Adding or changing field data

To add or change field data in the target table, press RETURN until the cursor is at the prompt for the field you want to edit. Then enter the new data, and press RETURN to complete the field edit.

If the new data is accepted, it is stored in the database and the cursor moves to the next prompt. If the data is not accepted, an error message displays at the bottom of the screen, and the database is not updated. Press RETURN to acknowledge the error, and try again.

If you start to enter data to a field and change your mind, press CTRL U to cancel the entry without changing the database. ENTER redisplays the old data to verify that it has not changed. However, if you press RETURN after entering the field data, CTRL U does not cancel the entry.

You can edit all fields in the target table this way, except TEXT and BINARY fields. ENTER can interface with a text editor and a binary editor to edit TEXT and BINARY fields.

Entering data in a text type field

To enter data to a TEXT field on a data entry screen, place the cursor on the first character of the TEXT field display. Then press CTRL V to start the TEXT field editor.

If the EDIT environment variable has been set, the specified editor is used. Otherwise, the default text editor, vi, is used.

Enter data in a binary type field

Before you can edit a BINARY field, you must set the BINEDIT environment variable to specify a binary editor. There is no default binary editor.

To edit a BINARY field on a data entry screen, place the cursor on the position reserved for the BINARY field. (This is the same position that would be represented by a B on the screen form in PAINT.) Then press CTRL V to start editing the field with the editor specified in BINEDIT.

Changing data in a secondary table

To change the data in a secondary table, you must define another ENTER screen with that secondary table as the target. Otherwise, use a host language program to manage the screen form.

15.3 Using ENTER with advanced field attributes (AFA)

You can use Advanced field attributes to specify default and legal values for individual fields. You can also specify a one-line information message, an error message, and a help file to be displayed for the field. For information on entering Advanced field attributes, see section 8.6.

The following describes the effects Advanced field attributes (AFA) have on ENTER screens:

Adding and modifying records with AFA

1. In add mode, if AFA defaults have been specified, all primary table screen fields are filled in with their default values when you enter the primary key value.
2. If no AFA defaults have been specified, the standard defaults are filled in as shown in Figure 15.3.

If the field data type is...	Then the AFA standard default is...
NUMERIC	0
FLOAT	0
AMOUNT	0.00
DATE	**/**/**
LDATE	**/**/**
TIME	00:00
STRING	(blank)
TEXT	n/a
BINARY	n/a

Figure 15.3 Standard AFA defaults

3. If the primary key field has an AFA default, press RETURN to move the cursor past the key field. All fields, including the primary key, are then filled in with their custom defaults.

4. If the table has no primary key, all fields with custom defaults are filled in when you select add mode or when you clear the screen after completing an add operation.

   You must enter data to at least one field to add the record. The record is not added if it only contains default values.

5. If an AFA default value has been set for a referencing field and you skip the field while adding a record, ENTER does not display the value from the referenced parent record. However, the reference is established, and the value will display when the record is recalled.

ENTER and AFA messages

Three types of AFA messages can be displayed on ENTER screens: FYI, ERROR, and HELP messages.

1. You can use AFA FYI messages to display the type of value to enter. This reduces the need to rely on the default value for a valid field entry.

   In both add and modify modes, as you move the cursor from field to field on the screen, the FYI message for the current field (if any) is displayed at the bottom of the screen.

2. You can use FYI messages to tell the user an error has been made.

   ENTER validates each field value you enter. If AFA legal values have been specified for the field, the field value entered must match the AFA legal values specified for the field. Otherwise, the value must match the standard default value for the field's data type.

   If you enter an invalid value and an AFA ERROR message has been specified for that field, the message displays at the bottom of the screen.

   The existing field value does not change until you enter a valid value,

3. If an AFA HELP file has been specified for a field, you can display the help text by pressing CTRL T or TAB when the cursor is on the corresponding screen field.

15.4 Using ENTER for query by forms

Query By Forms is the process of locating a record or set of records that you want to inquire, modify, or delete. The following describes how to locate records with Query By Forms.

As with data entry, query by forms is best illustrated by example. Refer to the *Unify DataServer/ELS Developer's Tutorial Manual* for a query by forms example.

Query by forms works in the following way.

After you select inquire, modify, or delete mode at the operation mode prompt on the ENTER screen, the following prompt displays:

```
Begin search [CTRL E], Clear field [CTRL Z], Exit [CTRL X]
```

This prompt indicates the default control key values for the begin search, clear field, and exit functions. You can change the control keys by editing your termcap file as discussed in section 6.1.

The control characters have the following functions:

CTRL E — Starts the search for the selection criteria you have entered.

If you realize that you have entered the wrong selection criteria and you want to interrupt the search while it is in progress, press the RUBOUT or DELETE key.

CTRL Z — Clears the current selection criteria. Use this when you have entered selection criteria to a field and decide that you don't want to search for that criteria.

CTRL X — Terminates the current ENTER screen and returns to the Menu Handler.

Unlike the two other control characters, CTRL X can be used whenever an ordinary character would be accepted. This gives you a quick way to exit an ENTER screen.

The simplest query is to locate a record by its primary key. Enter the key value, and ENTER can find the record without searching.

After you locate a record this way in modify mode, press RETURN to move the cursor to the field you want to edit, and update the record by entering new data.

Besides locating records by primary key, ENTER has two additional query capabilities: exact matching and inexact or generic matching.

Exact matching is when you fill in the field or fields on the screen form with exactly what you want the selected records to contain.

Inexact matching is when you enter expressions using combinations of values and special characters rather than exact values. Examples of inexact matches are numeric and date ranges, such as numbers from 1 to 100, or dates from 3/1/82 to 3/31/82; or substring matches, such as all the strings that contain the string "Smith".

Inexact matching on string fields

To specify an inexact match on string fields, enter a search expression containing special characters. ENTER uses these special characters to form patterns to be matched. The length of your search expression string can be two times the field length, plus two characters.

The following describes the special characters used to construct search expressions for inexact matching:

?	The *wildcard character*. The question mark matches a single character. For example, if you want to find all the Smiths, but don't know whether they are spelled "Smith" or "Smyth", you can use the specification Sm?th.
*	The *wildcard string*. The asterisk matches a string of characters of any length, including zero-length strings, or null strings.
[...]	A *character class.* The character class matches a single character that is a member of the class.

You can specify ranges of characters by separating two characters with a dash (-). For example, you can specify all upper case letters by the class:

[ABCDEFGHIJKLMNOPQRSTUVWXYZ]

or more conveniently, by [A-Z]. You can specify all upper and lower case letters by [a-zA-Z]. You can specify other classes similarly.

Inexact matching on non-string fields

You can also specify inexact matches on non-string fields with the following set of expressions. The values you supply are f1 and f2.

> f1	The greater than expression. All fields with values greater than the entered value match the expression.
<f1	The less than expression. All fields with values less than the entered value match the expression.
!f1	The logical not expression. All fields that do not match the entered value match the expression.
f1-f2	The range expression. All field values that match the entered values, or are between the entered values, match the expression. This is equivalent to >= f1 AND <=f2.
!f1-f2	This expression matches all field values that are outside the range of entered values. This is equivalent to <f1 OR>f2.

You can specify search expressions for any number of fields on the screen form. The search expression must begin in the first character position of the screen field. The result is to AND all the expressions together to produce a query.

Using queries to modify, delete, or report

After a set of records has been selected, the first record in the set displays on the screen form. The operation prompt displays the current valid operations you can choose. Which operations are valid depend on the current mode.

The six possible operations are as follows:

[N]EXT, [P]REVIOUS, [M]ODIFY, [D]ELETE, [R]EPORT, [S]TOP

[N]EXT, [P]REVIOUS, and [S]TOP display in all modes.

[M]ODIFY displays only in modify mode. [D]ELETE displays only in delete mode. And [R]EPORT displays in inquire mode if a formatting program is associated with the ENTER screen. The possible responses to the operation prompt are described as follows:

n	Displays the next record in the set of selected records. A RETURN also displays the next record.
p	Displays the previous record in the set of selected records.
m	Moves the cursor to the first screen field in the current record so you can modify the data. If someone else is modifying or deleting this record, you cannot modify the record.
d	Deletes the current record from the database. If someone else is modifying or deleting this record, you cannot delete the record.
r	Displays the ENTER Report Options Screen if formatting programs are associated with the current ENTER screen. Up to eight different formatting programs can be associated with a screen form.

Reports are generated for the current set of selected records. For more information about ENTER and reports, see ENTER report options screen later in this chapter.

s — Clears the current set of selected records and erases the screen so you can perform another query. A CTRL U also clears the selection and erases the screen. When records are selected using ENTER — query by forms, the selected count registers display below the operation prompt. The selected count registers are described as follows:

searched: nnn
: Shows the number of records (nnn) ENTER examined to satisfy the query.

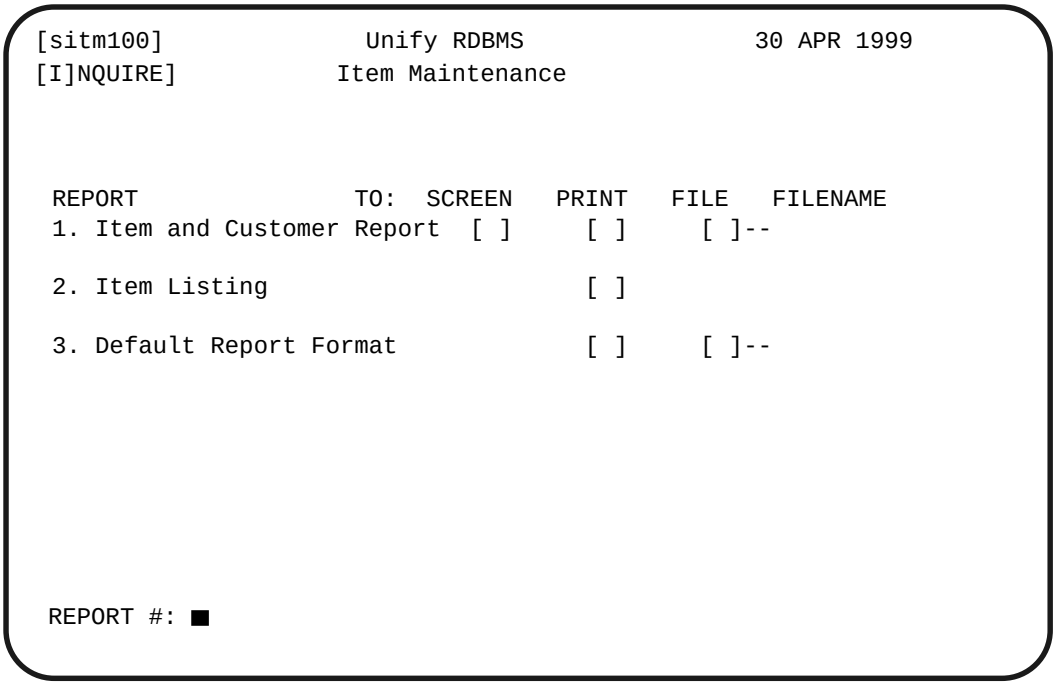
selected: nnn
: Shows the number of records (nnn) ENTER selected.

current: nnn
: Shows the number of the record (nnn) currently displayed on the screen. The number ranges from 1 to the number of selected records.

Enter report options screen

After you have selected a set of records using ENTER query by forms, you can use the ENTER report options screen to choose a report and specify where to send its output. The available reports and their valid output destinations are controlled by Register screen form with ENTER (earlier in this chapter).

In inquire mode, if you have registered a report with ENTER for the current ENTER screen, enter r at the operation prompt to display the Report options screen. The following is an example of an ENTER Report options screen:



The prompts on the Report options screen are as follows:

REPORT #: You can enter any listed report number to choose a report. RETURN defaults to the first report.

From this prompt, you can also escape to the operating system shell, or to a text editor by entering edit. This is useful when you want to modify formatting scripts with an editor, or you want to look at output that has been redirected to a file.

REPORT The title of the report to be produced when you select this option. This string derives from TITLE on the Register screen form with ENTER screen.

TO: Output can be directed to the following three destinations:

SCREEN The terminal screen.

PRINT The print spooler.

FILE A file with name FILENAME.

[ ] Enter an x between the brackets to select the corresponding output destination for the current report. Any combination of available destinations is valid.

-- When you choose a FILE output destination, you must enter the output filename after this prompt. If the filename exists, the following message displays:

File already exists, destroy it?

Respond with a Y or y to overwrite the file. Respond with a N or n to enter a new file name.

When you choose a report at the REPORT #: prompt, the requested report line is highlighted and the cursor moves to the first allowed output destination prompt. After you choose the appropriate destinations, return to the first output prompt with CTRL U or RETURN. Enter CTRL U to display the process mode prompt. If the chosen report has only one valid output destination, that option is automatically requested.

The process mode prompt then displays on the REPORT #: line as follows:

Proceed in [F]oreground, [B]ackground, [D]ebug or [C]ancel?

This prompt lets you choose the process mode for the report.

The option [D]ebug does not display if the nodebug mode (N) has been set with ENTER Screen Registration. The [B]ackground option is not allowed if SCREEN has been chosen as an output destination. The possible options are as follows:

f Process the report in foreground mode (while you wait).

b Process the report in background mode (while you work on something else).

d	Process the report in foreground mode, and display the report scripts, run time results, and error messages. Debug is useful because it displays the scripts used with the argument substitution.
c	Cancel the report request and return to the REPORT #: prompt.

Chapter 16: SQL-Query/DML language

This chapter describes the major Unify DataServer/ELS query language interface—SQL. SQL was developed at the IBM Research Center as a relational inquiry and data manipulation language based on an English keyword syntax. Its structure was refined and extensively tested to produce a language easy enough for nonspecialists to use, yet powerful enough for data processing professionals.

SQL is fast becoming the standard relational query language on all sizes of computers.

The Unify DataServer/ELS version of SQL is based on the language described by D.D. Chamberlin and his associates in "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control".

To adapt SQL to supermicros and the operating system environment, some changes have been made to the syntax. Most of the query and data manipulation features have been adopted without modification, which means that Unify DataServer/ELS SQL is the most powerful relational data language available on supermicros today.

All the examples in this chapter are based on a simple personnel database, containing employee, department, and taxes tables. Figure 16.1 shows the three database tables.

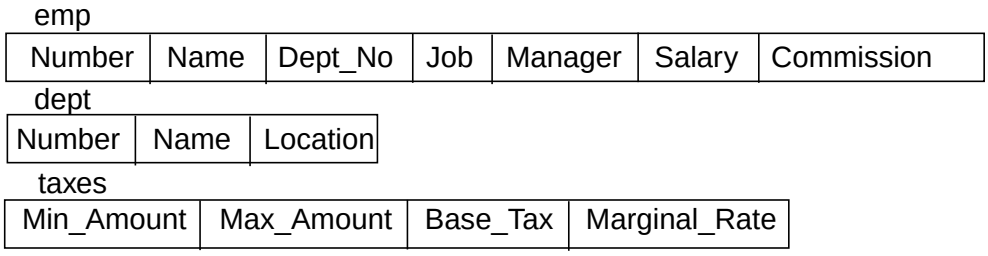


Figure 16.1 database Tables Used in SOL Examples

The emp table describes employees, giving the employee number, name, department number, job, manager number, monthly salary, and yearly commission. The dept table

describes departments, giving the department number, name, and location. The taxes table describes a personal income tax schedule, giving the minimum and maximum compensation amounts, base tax, and the marginal tax rate on income above the minimum amount.

16.1 Query facilities

A query is a question about your database, enabling you to retrieve or update database information. SQL lets you specify queries of varying degrees of complexity. An SQL query consists of statements and clauses.

Statements are commands. They are the verbs that specify what the query is to do; for example, "select," "update," "delete," or "insert"' records.

Clauses are the phrases that specify how a query should be done; for example, "select Emp_Name, Emp_Number," "from employees," or "order by Dept_No." Each clause is preceded by a keyword, which is either a statement or a modifier that further specifies how the query should be done.

Statements and clauses are made up of keywords. Keywords have specific meanings in SQL, and so cannot be used as database table or field names. For a list of the SQL reserved keywords, see SQL Reference, section 16.5.

Required and optional clauses

Some clauses are required, and some are optional.

The required SQL clauses are:

select	some data (a list of field names)
from	some place (a list of tables)

The optional SQL clauses are:

where	a condition (a true/false statement)
group by	some data (a list of field names)
having	a group condition (a true/false statement)
order by	some data (a list of field names)
into	a file

Identifying tables and fields in queries

Identify each table and field you want to use in selections and calculations by the table's short name and the database field's access name. The access name for a field is usually the *long* field name from the Unify DataServer/ELS database design.

However, for a field that references a combination (COMB) field, each of its implied fields has a separate access name. In this case, the implied field's access name is a compound of the referencing field's short database name and the short name of the component field it references in the COMB field.

A long field name need not be unique throughout the database, as a short field name must. A long name is just unique in a specific table.

If you give the same long name to different fields, and you want to use both fields in the same query, you must qualify each long field name with the name of the table. This is described in General Joins, later in this chapter.

The rules for valid long field names, referred to as field names or fields, are as follows:

1. Field names must begin with a letter. They can contain only upper or lower case letters, digits, and the underscore character (_).
2. Field names cannot contain spaces. SQL cannot distinguish a single field with a space from two separate fields.
3. Field names must be unique in a table,

Entering query statements

SQL lets you enter query statements in a free-form manner. This means that you can use extra spaces and carriage returns to make a query easier to read, without changing the query's meaning. All the queries in the following chapters exhibit this feature.

A query is ended by the slash character, /. When SQL reaches the slash character, it begins to evaluate what you have entered. If you have entered a valid query, the following message displays:

recognized query!

to let you know that your query is being processed.

If SQL cannot understand your query, it displays an error message that indicates what kind of error you have made. For a summary of error messages and suggested solutions, see Messages at the end of section 16.5. And for information on how to edit, save, and restart saved queries, see SQL Extensions, chapter 16.4.

When you want to exit SQL, enter end, and you return to the Menu Handler.

The following subsections describe each of the SQL clauses using example queries. However, because of the complexity of SQL, examples of every possible kind of query cannot be demonstrated. Therefore, refer to SQL Reference, chapter 16.5, for a complete explanation of formal query syntax.

Help features

Unify DataServer/ELS SQL has three levels of help to make learning and using SQL easier.

- Help about the general syntax of SQL
- Help about any specific keyword or keyword clause
- Help about the valid table and field names for the database you are using

To get help about the general syntax of SQL, enter:

```
sql> help
```

at the SQL prompt. SQL then lists the valid keywords about which you can obtain more information.

For more information about a specific keyword, enter help followed by the keyword. For example, to find out more about the select clause, enter:

```
sql> help
```

select at the SQL prompt. SQL displays an explanation of how to use the select clause.

Information is also available about the tables and fields of the database you are working with. To see a list of the valid short table names in the current database, enter:

```
sql> tables
```

at the SQL prompt. SQL displays the table names you can use with the from clause.

Similarly, to find out the field access names for any table, enter fields followed by the name of the table. For example, to list the fields in the emp table, enter:

```
sql> fields emp
```

at the SQL prompt. SQL displays the field access names for this table.

The following examples show the short table names and the field access names for the database used in this chapter:

```
sql> tables
emp       dept      taxes

sql> fields emp
```

NAME	TYPE	LENGTH
Number	INTEGER	4
Name	STRING	10
Dept_No	INTEGER	2
Job	STRING	10
Manager	INTEGER	4
Salary	AMOUNT	4
Commission	AMOUNT	4

```
sql> fields dept
```

NAME	TYPE	LENGTH
Number	INTEGER	2
Name	STRING	15
Location	STRING	15

```
sql> fields taxes
```

NAME	TYPE	LENGTH
Min_Amount	AMOUNT	5
Max_Amount	AMOUNT	5
Base_Tax	AMOUNT	5
Marginal_Rate	FLOAT	42

select...from CLAUSE

The simplest SQL query includes both a select clause and a from clause. The select clause lists the fields to be retrieved, and the from clause tells which table (or tables) the fields are to come from. The fields to be selected must be in the tables listed in the from clause.

EXAMPLE: Select all the fields for each record of the emp table. This displays the contents of all the emp records. The "*" is shorthand for all the fields.

```
sql> select *
sql> from emp/
recognized query!
```

Number	Name	Dept_No	Job	Manager	Salary	Commission
1000	Smith	50	salesrep	2100	1500.00	1000.00
2000	Jones	10	clerk	1300	900.00	0.00
1100	Whittaker	20	salesrep	2400	2500.00	500.00
2100	Reilly	30	salesrep	2400	2500.00	1500.00
1200	O'Neil	20	salesrep	1100	1500.00	150.00
2200	Dugan	40	salesrep	2400	1650.00	900.000
1300	Schmidt	60	programmer	2400	2500.00	0.00
2300	Klein	20	salesrep	1100	1500.00	0.00
1400	Scharf	10	clerk	2000	800.00	0.00
2400	Lee	10	president	2400	7500.00	0.00
1500	Otsaka	60	engineer	1300	1800.00	0.00
2500	Kawasaki	30	salesrep	2100	1800.00	1000.00
1600	Dupre	50	clerk	1000	800.00	0.00
2600	Bleriot	10	programmer	1300	1100.00	0.00

1700	Moehr	70	clerk	2400	950.00	0.00
2700	Colucci	40	salesrep	2200	2500.00	3000.00
1800	Amato	40	salesrep	2200	2000.000	750.00
2800	Fiorella	70	clerk	1700	800.00	0.00
1900	Brown	60	engineer	1300	2000.00	0.00

EXAMPLE: Select all the fields for each record of the dept table.

```
sql> select *
sql> from dept/
recognized query!
```

Number	Name	Location
10	Administration	Dallas
20	Eastern Sales	New York
30	Central Sales	Chicago
40	Western Sales	Los Angeles
50	Marketing	San Francisco
60	Research	Dallas
70	Finance	Dallas

EXAMPLE: Select all fields for each record of the taxes table.

```
sql> select *
sql> from taxes/
recognized query!
```

Min_Amount	Max_Amount	Base_Tax	Marginal_Rate
0.00	3400.00	0.00	0.00000
3400.00	5500.00	0.00	0.12000
5500.00	7600.00	252.000	0.14000
7600.00	11600.00	546.00	0.16000
11900.00	16000.00	1234.00	0.19000
16000.00	20200.00	2013.00	0.22000
20200.00	24600.00	2937.00	0.25000
24600.00	29900.00	4037.00	0.29000
29900.00	35200.00	5574.00	0.33000
35200.00	45800.00	7323.00	0.39000
45800.00	6000.00	11457.00	0.44000

```
6000.00    |   85600.00    17705.00 |       0.49000
85600.00   |   99999.00    30249.00 |       0.50000
```

When you use the "*" to select all the fields, the fields display in system-determined order. When you select the fields separately, you can control the sequence and number of fields.

EXAMPLE: List the employee number, Job, name, and salary for every employee.

```
sql> select Number, Job, Name, Salary
sql> from emp/
recognized query!
```

Number	Job	Name	Salary
1000	salesrep	Smith	1500.00
2000	clerk	Jones	900.00
1100	salesrep	Whittaker	2500.00
2100	salesrep	Reilly	2500.00
1200	salesrep	O’Neil	1500.00
2200	salesrep	Dugan	1650.00
1300	programmer	Schmidt	2500.00
2300	salesrep	Klein	1500.00
1400	clerk	Scharf	800.00
2400	president	Lee	7500.00
1500	engineer	Otsaka	1800.000
2500	salesrep	Kawasaki	1800.00
1600	clerk	Dupre	800.00
2600	programmer	Bleriot	1100.00
1700	clerk	Moehr	950.00
2700	salesrep	Colucci	2500.00
1800	salesrep	Amato	2000.00
2800	clerk	Fiorella	800.00
1900	engineer	Brown	2000.00

where CLAUSE

Because you rarely want to list the entire contents of a specific table, you can use the where clause to specify selection criteria. Selection criteria are the conditions that must be satisfied for a record to be selected.

The selection criteria in the where clause can consist of simple or complex expressions combining the following types of operators and notation:

- comparison operators
- logical operators
- set inclusion notation

The following examples illustrate how each type of operator or notation can be used in the where clause.

COMPARISON OPERATORS (=, ^=, <, >, <=, >=)

Comparison operators are used in a where clause to compare a field with a constant or expression, or with the results of another select clause. Comparison operators (sometimes called relational operators) compare or relate the expression on the left side of the operator with the expression on the right.

The where clause can also compare two fields with each other, using any of the standard comparison operators, =, ^=, <, <=, >, and >=. However, String fields can only be compared using the = and ^= operators.

NOTE: Do not insert a space between the two characters in the operators "=,>=, or <=.

Comparing Numbers

The following example illustrates comparing a field with a numeric constant.

EXAMPLE: List the name and location of department number 70.

```
sql> select Name, Location
sql> from dept
sql> where Number = 70/
```

```
recognized query!
Name|Location
__________|_______
Finance|Dallas
```

EXAMPLE: List the name, job, salary, and commission for employees whose commission exceeds their salary.

```
sql> select Name, Job, Salary, Commission
sql> from emp
sql> where Commission > Salary/
recognized query!

 Name     | Job       | Salary   | Commission
 _______  | _________ | ________ | __________
 Colucci  | salesrep  | 2500.00  | 3000.00
```

Comparing Strings

Suppose you want to find all the departments in Dallas. This requires comparing the Location field with the string constant, Dallas.

Unify DataServer/ELS SQL string constants can contain a combination of alphanumeric characters and the special characters: *, ?, and []. The special characters are described as follows:

? - The *wildcard character.* The question mark matches any single character. Thus, if you want to find all the Smiths and don't know if they are spelled "Smith" or "Smyth," you can use the specification Sm?th.

* - The *wildcard string.* The asterisk matches any string of characters of any length, including zero-length strings or *null strings.*

[...] - A *character class*. The character class matches any single character that is a member of the class. You can specify a range of characters by separating two characters with a dash (-).

For example, you can specify the range of all upper case letters by the class:

[ABCDEFGHIJKLMNOPQRSTUVWXYZ]

or more conveniently by [A-Z]. You can specify the range of all upper and lower case letters by [a-zA-Z]. You can construct other classes similarly.

NOTE: String constants must be enclosed in single quotes (') to distinguish them from field names.

EXAMPLE: List the names and locations of all departments in Dallas.

```
sql> select Name, Location
sql> from dept
sql> where Location = 'Dallas'/
recognized query!
```

Name	Location
Administration	Dallas
Research	Dallas
Finance	Dallas

EXAMPLE: List the names and jobs of all employees whose names begin with the letters A through M.

```
sql> select Name, Job
sql> from emp
sql> where Name = '[A-M]*'/
recognized query!
```

Name	Job
Jones	clerk
Dugan	salesrep
Klein	salesrep
Lee	president
Kawasaki	salesrep
Dupre	clerk
Bleriot	programmer
Moehr	clerk
Colucci	salesrep
Amato	salesrep
Fiorella	clerk
Brown	engineer

EXAMPLE: List the names and jobs of all employees whose names have 5 or less nonblank characters.

```
sql> select Name, Job
sql> from emp
sql> where Name = '????? '/
recognized query!
```

Name	Job
Smith	salesrep
Jones	clerk
Dugan	salesrep
Klein	salesrep
Lee	president
Dupre	clerk
Moehr	clerk
Amato	salesrep
Brown	engineer

Logical Operators (and, or, not)

The where clause can also contain complex Boolean expressions composed of selection criteria connected by *and* and *or* operators. Precedence can be established by using square brackets [ ].

You can use the standard Boolean operators *and* and *or* to connect simple comparisons to form complex expressions. Use the not operator to negate an expression.

SQL evaluates expressions containing logical operators in the following order of precedence: not, and, or. If an operator appears more than once in a query, SQL evaluates the expressions containing it from left to right. If you need to change the order of evaluation, use square brackets [] to indicate which part of the expression to evaluate first.

EXAMPLE: List the name, job, salary, and department number for the employees in department 10 who are either clerks or earn less than or equal to $1200.

```
sql> select Name, Job, Salary, Dept_No
sql> from emp
sql> where Dept_No = 10 and sql> [Job = 'clerk' or Salary <= 1200]/
recognized query!
```

Name	Job	Salary	Dept_No
Jones	clerk	900.00	10
Scharf	clerk	800.00	10
Bleriot	programmer	1100.00	10

Frequently, you want to select records based on a range of values. With the standard comparison operators, this might be awkward. For example, if you want to find employees with monthly salaries between $1000 and $2000, you might use the following expression:

salary >= 1000 and salary <= 2000

SQL has a between operator that lets you express this easier.

EXAMPLE: List the name, salary, and job of employees earning between $1500 and $2000, inclusive.

```
sql> select Name, Salary, Job
sql> from emp
sql> where Salary between 1500 and 2000/
recognized query!
```

Name	Salary	Job
Smith	1500.00	salesrep
O’Neil	1500.00	salesrep
Dugan	1650.00	salesrep
Klein	1500.00	salesrep
Otsaka	1800.00	engineer
Kawasaki	1800.00	salesrep
Amato	2000.00	salesrep
Brown	2000.00	engineer

Boolean expressions can be negated in whole or in part to select records that do not match a specified criteria.

For instance, suppose the research and development department has such employees as engineers, programmers, clerks, and typists. Using either the not operator or the ^= operator, you can select all employees except engineers without listing each job title.

The following two examples compare the use of the ^= comparison operator and the not logical operator.

EXAMPLE: List the department number, name, job, and salary of everyone in department 60 who is not an engineer. Use the ^= operator.

```
sql>select Dept_N0, Name, Job, Salary
sql> from emp
sql> where Dept_No = 60 and
sql> Job ^= 'engineer' /
recognized query!
```

Dept_No	Name	Job	Salary
60	Schmidt	programmer	2500.00

EXAMPLE: List the name, job, and salary of all employees who are not salesreps or who make less than $2000. Use the not operator. The square brackets indicate that the operator applies to the whole expression.

```
sql> select Name, Job, Salary
sql> from emp
sql> where not [Job = 'salesrep' or Salary >= 2000] /
recognized query!
```

Name	Job	Salary
Jones	clerk	900.00
Scharf	Clerk	800.00
Otsaka	engineer	1800.00
Dupre	clerk	800.00
Bleriot	programmer	1100.00
Moehr	clerk	950.00
Fiorella	clerk	800.00

In many queries, you may want to compare a field to a list of values, not just a single value.

For example, suppose you want to select all the employees in departments 10, 20, 30, and 40. With the standard operators, this becomes a sequence of equalities connected by ors, such as the following:

Dept_No = 10 or Dept_No = 20 or Dept_No = 30 ...

As with ranges, this can be awkward.

SQL has a set inclusion notation to make this kind of query easier. To use set inclusion notation, enclose a series of values, separated by commas, in angle brackets < >.

EXAMPLE: List the name, job, and department number for employees in departments 20, 30, and 40.

```
sql> select Name, Job, Dept_No
sql> from emp
sql> where Dept_No in <20, 30, 40> /
recognized query!
```

Name	Job	Dept_No
Whittaker	salesrep	20
Reilly	salesrep	30
O'Neil	salesrep	20
Dugan	salesrep	40
Klein	salesrep	20
Kawasaki	salesrep	30
Colucci	salesrep	40
Amato	salesrep	40

Set inclusion notation can also be used to group and expressions together. This is useful if you want to select based on a combination of expressions.

For example, you can select all clerks in department 10 and all programmers in department 60. Using standard notation, this is expressed as:

[Job = 'clerk' and Dept_No = 10]

OR

[Job = 'programmer' and Dept_No = 60]

Set inclusion notation can be used to express this kind of statement more easily.

EXAMPLE: List the name, job, salary, and department number of the clerks in department 10 and the programmers in department 60. Use parentheses to group the set of constants.

```
sql> select Name, Job, Salary, Dept_No
sql> from emp
sql> where <Job, Dept_No> is in ( <'clerk', 10>,
sql>                               <'programmer", 60> ) /
recognized query!

 Name    | Job        | Salary   | Dept_No
---------+------------+----------+---------
 Jones   | clerk      |  900.00  |      10
 Schmidt | programmer | 2500.00  |      60
 Scharf  | clerk      |  800.00  |      10
```

Sets of constants such as:

<'clerk', 10>

are called *literal tuples* in relational terminology. A literal tuple is a group of constants describing values for specific fields in a record. The fields must be identified by preceding the tuples with a field specification list, as in the previous example.

For another use of literal tuples—inserting records into the database—see Data Manipulation Language Facilities, section 16.2.

unique KEYWORD

If a query doesn't select a primary key field from one of the tables, the query can produce rows that duplicate each other. Because only the primary key must be unique, other fields can contain duplicate information.

Usually, these duplicates are not necessary. The unique keyword lets you eliminate duplicate information in a query result. The unique keyword also causes the query results to be sorted so duplicate information can be grouped and easily deleted.

For example, the emp table has many clerk and programmer records. If you list the job titles by just selecting all the records in the table, several titles will be duplicates. You can use the unique keyword so the query selects only one record for each Job title.

EXAMPLE: List the different job titles in the company.

sql> select unique Job
sql> from emp/
recognized query!

```
Job
---------
clerk
engineer
president
programmer
salesrep
```

The following two examples demonstrate the differences in query results when the unique keyword is omitted and when it is used.

EXAMPLE: List the department number and job classifications of employees in departments 10, 20, and 30, or employees who make more than $2000. Note that this query produces duplicate information.

```
sql> select Dept_No, Job
sql> from emp
sql> where Dept_No in <10, 20, 30> or
```

```
sql>       Salary > 2000 /
recognized query!

 Dept_No | Job
      20 | salesrep
      30 | salesrep
      20 | salesrep
      60 | programmer
      20 | salesrep
      10 | clerk
      10 | president
      30 | salesrep
      10 | programmer
      40 | salesrep
```

EXAMPLE: Repeat the previous query, but eliminate the duplicates. Again, the results are sorted.

```
sql> select unique Dept_No, Job
sql> from emp
sql> where Dept_No in <10, 20, 30> or
sql>       Salary > 2000/
recognized query!

 Dept_No | Job
      10 | clerk
      10 | president
      10 | programmer
      20 | salesrep
      30 | salesrep
      40 | salesrep
      60 | programmer
```

Arithmetic expressions

Arithmetic expressions let you calculate values using the standard arithmetic operators +, -, *, and /. Arithmetic expressions can be used wherever a field is allowed in select, where, and having clauses.

Both constants and fields can be used in arithmetic expressions, which are allowed on fields of types NUMERIC, FLOAT, AMOUNT, DATE, LDATE, and TIME.

NOTE: The subtraction operator (-) must be surrounded by spaces to distinguish it from the minus sign.

EXAMPLE: List the name, job, and total salary plus commission for employees in departments 20, 30, and 40. These are the sales departments.

```
sql> select Name, Job, Salary + Commission
sql> from emp
sql> where Dept_No in <20, 30, 40> /
recognized query!
```

Name	Job	Salary+Commission
Whittaker	salesrep	3000.00
Reilly	salesrep	4000.00
O'Neil	salesrep	1650.00
Dugan	salesrep	2550.000
Klein	salesrep	1500.00
Kawasaki	salesrep	2800.00
Colucci	salesrep	5500.00
Amato	salesrep	2750.00

EXAMPLE: List the name, salary, commission earned, and earning potential for all salesreps who aren't earning commissions of at least $100 plus half their monthly salary.

```
sql> select Name, Salary, Commission,  (Salary * .5) + 100
sql> from emp
sql> where Commission < (Salary * .5) + 100 and
sql>        Job = 'salesrep*' /
recognized query!
```

Name	Salary	Commission	(Salary* .5)+100
Whittaker	2500.00	500.00	1350.00
O'Neil	1500.00	150.00	850.00
Dugan	1650.00	900.00	925.00
Klein	1500.00	0.00	850.00

```
 Name       | Salary    | Commission | (Salary* .5)+100
 ___________|___________|____________|_________________
 Amato      |  2000.00  |     750.00 |          1100.00
```

EXAMPLE: Suppose you make $3000 per month and earn commissions of $5000 per year. Compute your total compensation, base tax amount, and marginal tax amount, using the data from taxes.

```
sql> select (3000*12)+5000, Base_Tax,
sql>        (((3000*12)+5000) -Min_Amount)*Marginal_Rate
sql> from taxes
sql> where (3000*12)+5000 between Min_Amount and Max_Amount/
recognized query!

(3000*12) +5000| Base_Tax| ( ( (3000*12) +5000) -Min_Amount) *Marginal_Rate
_______________|_________|_____________________________________________
41000.00000    | 7323.00 |                                      2262.00
```

Aggregate functions

SQL has five built-in aggregate functions to calculate grouped items in a query. The functions are count(*), min, max, sum, and avg.

Aggregate functions are only valid in select or having clauses. You cannot use an aggregate function directly in a where clause, although you can achieve the same effect by using a nested query. This is detailed in Nested Queries, later in this chapter.

An aggregate function applies to a group of records with a common characteristic; for example, the average salary of employees in department 10. In this case, the common characteristic is the department number.

When you use an aggregate function in the select clause, you must select fields that remain constant for each member of the group. To select an employee name with the average salary for a department is meaningless, because each employee has a different name. If you want to list the names of employees who make the average salary, you must use a nested query.

Aggregate functions are usually used with the group by clause. A group by clause lets you partition the selected records into groups for which the indicated aggregate functions are computed. If you specify no explicit group by, then any aggregate functions apply to the entire set of selected records.

EXAMPLE: Compute and list the total number of employees in department 10.

```
sql>select count(*)
sql>from emp
sql>where Dept_No=10/
recognized query!
count(*)
_______________
              4
```

EXAMPLE: Compute and list the minimum and maximum salaries throughout the entire company. Remember, if you want to list the names of the employees who make these salaries, you must use a nested query.

```
sql> select min(Salary), max(Salary)
sql> from emp/
recognized query!
min(salary)|max(Salary)
___________|_________
        800.00| 7500.00
```

EXAMPLE: List the job and average salary plus commission for all salesreps. The job can be listed in this example because all the selected records have the same job: salesrep.

```
sql> select Job, avg(Salary + Commission)
sql> from emp
sql> where Job = 'salesrep*' /
recognized query!
Job|avg(Salary+Commission)
_______|_____________________
salesrep|                      2916.67
```

EXAMPLE: Compute how much the company is paying engineers and programmers yearly. This requires selecting all engineers and programmers, multiplying their monthly salaries by 12, adding in their commissions (presumably zero), and adding the total.

```
sql> select sum((Salary*l2) + Commission)
sql> from emp
sql> where Job = 'engineer*' or Job = 'programmer*' /
recognized query!
sum( (Salary*12)+Commission)
____________________________
                  88800.00
```

order by CLAUSE

The results of the previous queries are in SQL-determined order. Although the unique operator sorts the output, you cannot control the order of output.

The order by clause lets you specify the sequence of the rows that result from a query. You can specify ascending order (A-Z) using asc or descending order (Z-A) using desc. The default sort sequence is ascending, with STRING fields sorted alphabetically.

EXAMPLE: List every employee's number, name, and job, sorted by employee number.

```
sql> select Number, Name, Job
sql> from emp
sql> order by Number/
recognized query!
```

Number	Name	Job
1000	Smith	salesrep
1100	Whittaker	salesrep
1200	O’Neil	salesrep
1300	Schmidt	programmer
1400	Scharf	clerk
1500	Otsaka	engineer
1600	Dupre	clerk
1700	Moehr	clerk
1800	Amato	salesrep
1900	Brown	engineer
2000	Jones	clerk
2100	Reilly	salesrep
2200	Dugan	salesrep
2300	Klein	salesrep
2400	Lee	president
2500	Kawasaki	salesrep
2600	Bleriot	programmer
1700	Colucci	salesrep
2800	Fiorella	clerk

You can sort more than one field and specify the direction for each sort using ascending (asc) or descending (desc).

EXAMPLE: List every employee's department number, name, and job by ascending name in descending department number.

```
sql>select Dept_No, Name, Job
sql>from emp
sql>order by Dept_No desc, Name asc/
recognized query!
```

Dept_No	Name	Job
70	Fiorella	clerk
70	Moehr	clerk
60	Brown	engineer
60	Otsaka	engineer
60	Schmidt	programmer
50	Dupre	clerk
50	Smith	salesrep
40	Amato	salesrep
40	Colucci	salesrep
40	Dugan	salesrep
30	Kawasaki	salesrep
30	Reilly	salesrep
20	Klein	salesrep
20	O'Neil	salesrep
20	Whittakerr	salesrep
10	Bleriot	programmer
10	Jones	clerk
10	Lee	president
10	Scharf	clerk

group by CLAUSE

The group by clause arranges rows that contain the same values together into groups. Use this clause when computing aggregate functions for groups of rows with common characteristics.

Each field specified in the select statement must also be specified in the group by clause; if not, the results of your query will be incorrect or misleading. The effect of a group by clause is to sort the selected rows by the indicated fields, then perform the aggregate functions at each level break. The output of this query is sorted also.

Use the group by clause only for queries that contain aggregate functions.

See how payroll is being allocated to the various departments.

EXAMPLE: For each department, list the department number, employee count, and total salary plus commission.

```
sql> select Dept_No, count(*), sum(Salary + commission)
sql> from emp
sql> group by Dept_No/
recognized query!
```

Dept_No	count(*)	sum(Salary+Commission)
10	4	10300.000
20	3	6150.000
30	2	6800.00
40	3	10800.00
50	2	3300.00
60	3	6300.00
70	2	1750.00

You can use more than one field in a group by clause, resulting in more divisions in the calculation.

Find the average salary by job title in departments 10, 20, and 30. This tells you what department pays the highest wage for a given job.

Find the average salary by job title in departments 10, 20, and 30. This tells you what department pays the highest wage for a given job.

EXAMPLE: List the department number, job, employee count, and the average salary for each job title by department for departments 10, 20, and 30.

```
sql.select Dept_No, Job, count(*), avg(Salary))
sql>from emp
```

```
sql>where Dept_No is in ,10, 20, 30>
sql>group by Dept_No, Job/
recognized query!
```

Dept_No	Job	count(*)	avg(Salary)
10	president	1	7500.00
10	programmer	1	1100.00
20	salesrep	3	1833.33
30	salesrep	2	2150.00

EXAMPLE: Compute the average yearly amount paid salesreps in each department. The yearly amount is the monthly salary times twelve, plus the commission. Also list the department number and salesrep count in each department.

```
sql>select Dept_No, count(*), avg(Salary*12)+Commission)
sql>from emp
sql>where Job+"salesrep*"
sql>group by Dept_No/
recognized query!
```

Dept_No	count(*)	avg(Salary*12)+Commission
20	3	22216.67
30	2	27050.00
40	3	26150.00
50	1	19000.00

You can also apply an aggregate function to the result of another aggregate function. This lets you compute such items as the maximum average or the average count. When used this way, an aggregate function requites a group by clause. SQL computes the result as follows:

1. All qualifying records are selected using the where clause, if any.
2. The selected records are sorted according to the fields in the group by clause.
3. The inner aggregate function is computed.
4. The outer function is then applied to the results.

Because the second level of computation removes all identity from the groups, you must use a nested query to list fields other than the function result.

EXAMPLE: List the average department size in the company.

```
sql> select avg(count(*))
sql> from emp
sql> group by Dept_No/
recognized query!
avg(count(*) )
_______________
        2.71429
```

EXAMPLE: List the maximum average monthly salary for all jobs, except "president." Note that you must use a nested query to see what job this is.

```
sql> select max(avg(salary))
sql> from emp
sql> where Job ^= 'president'
sql> group by Job/
recognized query!
  Max(avg(Salary) )
_____________________
               1938.89
```

Nested queries

Nested queries enable you to ask questions that cannot be answered using the previous SQL queries. Nesting lets you use the results of one query as input to another, so you can use the results of one question to answer another.

Suppose for example, you want to find the name of the employee who earns the most. Using the capabilities presented so far, you could find the maximum amount made by any employee as follows:

```
select max (Salary + Commission) from emp/
```

Or, you could find the employee who earns any constant amount, such as over $2500:

```
select Name from emp where Salary + Commission > 2500/
```

But the only way to determine who earns the most is to perform the first query, then use the resulting value of salary plus commission as the constant value in the second query. You can do this with nesting.

EXAMPLE: Find the name and job of the employee who earns the maximum salary plus commission.

```
sql> select Name, Job
sql> from emp
sql> where Salary + Commission =
sql>       select max (Salary + Commission)
sql>        from emp/ recognized query!
Name  |Job
______|_________
Lee   |president
```

This query works by evaluating the inner query to obtain a value for the maximum salary plus commission. This value is then used as a constant by the outer query, which finds the employee (s) who earn that amount.

You can also use the previously-described set inclusion notation in nested queries. For example, suppose an employee not in department 10 suspects that everyone in that department earns more because department 10 is company headquarters. A nested query using set inclusion can be used to confirm or deny this.

EXAMPLE: List the name, job, and salary of employees not in department 10 who have the same job and salary as employees in department 10.

```
sql> select Name, Job, Salary
sql> from emp
sql> where Dept_No ^= 10
sql>           select Job, Salary
sql>           from emp
sql>           where Dept_No = 10/
recognized query!
Name     |Job     | Salary
_______  |________|________
Dupre    |clerk   |  800.00
Fiorella |clerk   |  800.00
```

You can nest queries to five levels, not just the two described so far. The following four-level query finds the person with the second highest earnings. The query finds

the name of the person with the maximum earnings and then finds the maximum earnings of the remaining employees.

This query illustrates how you can compare an expression with the results of a nested query. To understand nested queries, you should start with the innermost query and work your way out.

EXAMPLE: List the department number, name, job, and salary plus commission of the person with the second highest earnings.

```
sql> select Dept_No, Name, Job, Salary + Commission
sql> from emp
sql> where Salary + Commission =
sql>       select max(Salary + Commission)
sql>       from emp
sql>             select Name
sql>             from emp
sql>             where Salary + Commission =
sql>                   select max(Salary + Commission)
sql>                   from emp/
recognized query!
Dept_No |Name    |Job     | Salary+Commission
________|_________| ______| _________________
40      |Colucci |salesrep|            5500.00
```

You can include nested queries in more complex expressions. In this case, the inner query must end with a semicolon (;) so SQL can determine where the nested query ends and the rest of the Boolean expression begins.

The following query compares the top salesrep in the company with all salesreps in department 20.

EXAMPLE: List the department number, name, job, total earnings, and commission for the salesrep with the maximum total earnings, and for all the salesreps in department 20. Sort the result by salary within commission in descending order, so the high earners come first.

```
sql> select Dept_No, Name, Job, Salary + Commission, Commission
sql> from emp
sql> where Salary + commission =
sql>        select max (Salary + Commission)
sql>       from emp
```

```
sql>       where Job = 'salesrep';
sql> order by commission desc. Salary desc/
recognized query!
Dept_No|Name       |Job       |Salary+Commission  |Commission
_______|___________|__________|___________________|___________
  40   |Colucci    |salesrep  |       5500.00     |     3000.00
  20   |Whittaker  |salesrep  |       3000.00     |      500.00
  20   |O'Neil     |salesre   |       1650.00     |      150.00
  20   |Klein      |salesrep  |       1500.00     |        0.00
```

having CLAUSE

The having clause lets you select some groups formed by a previous group by clause and reject others. This is based on the results of another selection using one or more aggregate functions. It is like using an aggregate function in a where clause, which you cannot do.

For example, you can use the having clause to select the departments where the average salary is over $2000.

EXAMPLE: List the department number and average salary for departments having an average salary over $2000.

```
sql> select Dept_No, avg(Salary)
sql> from emp
sql> group by Dept_No
sql> having avg(Salary) > 2000/
recognized query!
Dept_No|     avg(Salary)
_______|_____________
   10 |      2575.00
   30 |      2150.00
   40 |      2050.00
   60 |      2100.00
```

When a query contains both a having and a where clause, SQL evaluates the query as follows:

1. The where clause is applied to select qualifying records.
2. The groups indicated by the group by clause are formed.
3. The having clause is applied to select qualifying groups.

EXAMPLE: List the department number and number of salesreps for each department with more than two salesreps.

```
sql> select Dept_No, count (*)
sql> from emp
sql> where Job = 'salesrep'
sql> group by Dept_No
sql> having count (*) > 2/
recognized query!
Dept_No|  count (*)
_______|___________
     20|         3
     40|          3
```

You have previously selected individual salesreps whose commissions are below par. Using a having clause that compares the results of two aggregate functions for each department, you can find the departments where the average commissions are above or below par. This is useful to confirm how the sales managers are performing.

EXAMPLE: List the department number, average salary, average commission, and target commission level (equal to half the salary plus $100) for departments whose average commission is at or above the target level.

```
sql> select Dept_No,avg(Salary),avg(Comaission),
sql>                 avg((Salary*0.5) + 100)
sql> from emp
sql> where Job = 'salesrep'
sql> group by Dept_No
sql> having avg(Commission) >= avg((Salary*0.5) + 100) /
recognized query!
Dept_No  | avg(Salary)|avg(Commission)| avg((Salary*0.5)+100)
_________|____________|_______________|_____________________
  30     |    2150.00 |      1250.00  |           1175.00
  40     |    2050.00 |      1550.00  |           1125.00
  50     |    1500.00 |      1000.00  |            850.00
```

You can use nested queries in the having clause. SQL evaluates a query nested in a having clause the same as a query nested in a where clause. This query lets you ask such questions as "what departments have an average salary less than the average for all departments?"

EXAMPLE: List the department number and average salary for departments with an average salary less than the average for all departments.

```
sql> select Dept_No, avg(Salary)
sql> from emp
sql> group by Dept_No
sql> having avg(Salary)
sql>        select avg(Salary)
sql>               from emp/
recognized query!
Dept_No| avg(Salary)
_______|_________
   20  | 1833.33
   50  | 1150.00
   70  |  875.00
```

You can use nested queries in both the where and having clauses at the same time. The following query finds the employees who work in the department with the highest average salary. SQL evaluates this query as follows:

1. The maximum average salary for all departments is computed.
2. The department with that average salary is located.
3. All the employees in that department are located.

EXAMPLE: List the name, job, and salary for employees in the department with the highest average salary.

```
sql> select Name, Job, Salary
sql> from emp
sql> where Dept_No =
sql>       select Dept_No
sql>       from emp
sql>       group by Dept_No
sql>       having avg(Salary) =
sql>              select max(avg(Salary))
sql>              from emp
sql>              group by Dept_No/
recognized query!
Name       |Job         |  Salary
___________|____________|__________
Jones      |clerk       |  900.00
Scharf     |clerk       |  800.00
```

```
Lee          |president  | 7500.00
Bleriot      |programmer | 1100.00
```

Join queries

The previous query examples have involved only a single table. However, you can list fields from a number of tables in a single SQL query.

Queries that list fields from several tables are called join queries because they combine, or join, the tables. The tables to be joined in the query are listed in the from clause, in any order.

In join queries, SQL determines the most efficient method of performing the selection and qualification. The following examples illustrate various methods of joining tables.

General Join

A join query forms the Cartesian product of the selected tables.

A Cartesian product is formed when two tables are joined so that every record in the first table is joined to every record of the second table. For example, if the first table has 100 records and the second table has 200 records, their Cartesian product would contain 20,000 records.

Cartesian Product Example

The Cartesian product is best understood by an example. Consider three tables. A, B, and C. Table C is the Cartesian product of A and B. It contains every possible combination of the records in tables A and B; i.e., the first record in A with all three records in B, and the second record in A with all three records in B

Figure 16.2 illustrates how A and B combine to form C.

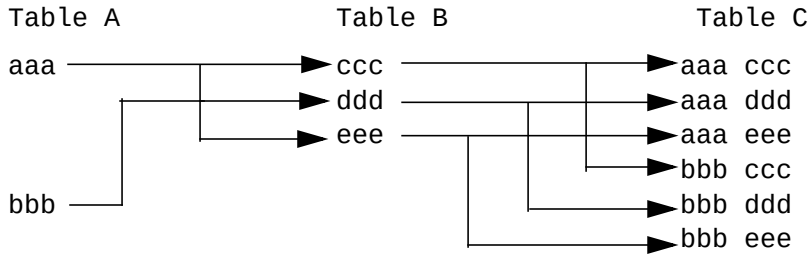


Figure 16.2 Cartesian Product of A and B is C

In this example, table A has two records, table B has three records, and the Cartesian product, table C, has six records.

When tables contain many records, performing a join that forms the Cartesian product and then selects the records takes much too long and lists a lot of redundant information. For example, suppose you want to join two tables, one with 5,000 records and one with 50,000 records. The Cartesian product of these two tables is 250,000,000 records. This would take days to process.

Unify DataServer/ELS SQL, however, can perform joins and selections on files of this size in seconds. To do this, each query takes advantage of the available access methods, including hashing, explicit relationships, and B-tree indexes.

Multiple access methods improve the performance of your database application by adding or changing access methods so that forming the Cartesian product is not necessary.

Joining Tables, without the where Clause

A join query without a where clause lists every record in the Cartesian product of the joined tables.

The following query illustrates a Cartesian product with no selection. You would probably never use such a query, but it is included so you can see what happens.

NOTE: If a field name is not unique, it must be preceded by the name of the table. Thus, to list the employee name, you must use emp.Name, because the dept table also has a field called Name.

EXAMPLE: List the employee name and department location for the Cartesian product of the employee and department tables.

```
sql> select emp .Name, Location
sql> from emp, dept/
recognized query!
emp. Name   | Location
____________|_______________
Smith       |Dallas
Smith       |New York
Smith       |Chicago
Smith       |Los Angles
Smith       |San Francisco
Smith       |Dallas
Smith       |Dallas
Jones       |Dallas
Jones       |New York
Jones       |Chicago
Jones       |Los Angeles
Jones       |San Francisco
Jones       |Dallas
Jones       |Dallas
Whittaker   |Dallas
Whittaker   |New York
Whittaker   |Chicago
Whittaker   |Los Angeles
Whittaker   |San Francisco
Whittaker   |Dallas
Whittaker   |Dallas
   "        |   "
   "        |   "
   "        |   "
```

The entire result is not listed, because it is too long. Note that the first employee is listed with the location of every department, and so on. Of course, this is not very useful information.

Joining Tables, with the where Clause

The real power of a relational DBMS is the ability to qualify the results of a join. The where clause lets you select a subset of the records from the Cartesian product. For example, you can select records where the department number in the emp table is the same as the department number in the dept table.

There are two factors to consider with this type of query:

1. The fields Name and Number must be preceded with table names because both emp and dept have fields with these names. The field Dept_No does not require qualification, because it only occurs in the emp table.
2. The expression dept.* is used to select all the fields in the dept table, without listing each one separately.

EXAMPLE: List the employee name and all the fields from the department where the employee works.

```
sql> select emp .Name, dept.*
sql> from emp, dept
sql> where Dept No = dept.Number/
recognized query!
emp. Name   | Number    | Name                |Location
____________|___________|_____________________|_________
Smith       |      50   |Marketing            |San Francisco
Jones       |      10   |Administration       |Dallas
Whittaker   |      20   |Eastern Sales        |New York
Reilly      |      30   |Central Sales        |Chicago
O'Neil      |      20   |Eastern Sales        |New York
Dugan       |      40   |Western Sales        |Los Angeles
Schmidt     |      60   |Research             |Dallas
Klein       |      20   |Eastern Sales        |New York
Scharf      |      10   |Administration       |Dallas
Lee         |      10   |Administration       |Dallas
Otsaka      |      60   |Research             |Dallas
Kawasaki    |      30   |Central Sales        |Chicago
Dupre       |      50   |Marketing            |San Francisco
Bleriot     |      10   |Administration       |Dallas
Moehr       |      70   |Finance              |Dallas
Colucci     |      40   |Western Sales        |Los Angeles
Amato       |      40   |Western Sales        |Los Angeles
Fiorella    |      70   |Finance              |Dallas
Brown       |      60   |Research             |Dallas
```

The where clause in a join query can contain expressions that use fields from any of the tables involved in the join. The expression is not limited to an equality, as the following query shows.

EXAMPLE: List the employee name and yearly salary from the emp table, and the amounts that the yearly salary falls between in the taxes table.

```
sql> select Name, Salary*l2, Min_Amount, Max Amount
sql> from emp, taxes
sql> where Salary*l2 between Min_Amount and Max_Amount/
recognized query!
Name         | Salary*l2  | Min_Amount   |  Max_Amount
_____________|____________|______________|_____________
Smith        |   18000.00 |   16000.00   |   20200.00
Jones        |   10800.00 |    7600.00   |   11600.00
Whittaker    |   30000.00 |   29900.00   |   35200.00
Reilly       |   30000.00 |   29900.00   |   35200.00
O'Neil       |   18000.00 |   16000.00   |   20200.00
Dugan        |   19800.00 |   16000.00   |   20200.00
Schmidt      |   30000.00 |   29900.00   |   35200.00
Klein        |   18000.00 |   16000.00   |   20200.00
Scharf       |    9600.00 |    7600.00   |   11600.00
Lee          |   90000.00 |   85600.00   |   99999.00
Otsaka       |   21600.00 |   20200.00   |   24600.00
Kawasaki     |   21600.00 |   20200.00   |   24600.00
Dupre        |    9600.00 |    7600.00   |   11600.00
Bleriot      |   13200.00 |   11900.00   |   16000.00
Moehr        |   11400.00 |    7600.00   |   11600.00
Colucci      |   30000.00 |   29900.00   |   35200.00
Amato        |   24000.00 |   20200.00   |   24600.00
Fiorella     |    9600.00 |    7600.00   |   11600.00
Brown        |   24000.00 |   20200.00   |    24600.00
```

Now that you have seen how to find the proper tax bracket, you can compute the tax liability for each employee. There is no limit to the number of tables you can join, so you can list the employee's name, location, total yearly compensation, and tax amount all in the same query.

The expression to calculate the tax amount is comparatively long, so the column heading has been edited to fit the result on the page.

EXAMPLE: List each employee's name, location, total yearly compensation, and tax bill.

```
sql> select emp.Name, Location, (Salary*l2)
sql> Base_Tax + (((Salary*12) + Commission)
sql> * Marginal_Rate
sql> from emp, taxes, dept
sql> where (Salary*12) + commission between
sql>       Min_Amount and Max_Amount
```

```
sql>       and Dept_No = dept.Number/
recognized query!

emp.Name   |Location       |(Salary*l2)+Commission  |Tax_Amount
___________|_______________|________________________|___________
Smith      |San Francisco  |    19000.00            |    2673.00
Jones      |Dallas         |    10800.00            |    1058.00
Whittaker  |New York       |    30500.00            |    5772.00
Reilly     |Chicago        |    31500.00            |    6102.00
O'Neil     |New York       |    18150.00            |    2486.00
Dugan      |Los Angeles    |    20700.00            |    3062.00
Schmidt    |Dallas         |    30000.00            |    5607.00
Klein      |New York       |    18000.00            |    2453.00
Scharf     |Dallas         |     9600.00            |     866.00
Lee        |Dallas         |    90000.00            |   32449.00
Otsaka     |Dallas         |    21600.00            |    3287.00
Kawasaki   |Chicago        |    22600.00            |    3537.00
Dupre      |San Francisco  |     9600.00            |     866.00
Bleriot    |Dallas         |    13200.00            |    1481.00
Moehr      |Dallas         |    11400.00            |    1154.00
Colucci    |Los Angeles    |    33000.00            |    6597.00
Amato      |Los Angeles    |    24750.00            |    4080.50
Fiorella   |Dallas         |     9600.00            |     866.00
Brown      |Dallas         |    24000.00            |    3887.00
```

Self Join

A self join is a query that joins a table with itself as if it were two separate tables.

For example, in the sample database, the emp table contains a field that indicates the employee's manager. This is merely the manager's employee number. Employee number 2100, Reilly, is the manager of Smith and Kawasaki. You can join the emp table to itself, using the employee's number and the manager's number.

The best way to understand this type of query is to imagine you have two copies of the emp table: one with employees and one with managers. SQL doesn't make a copy, but achieves the same effect by letting you give a table a temporary name. You can join temporary tables like any other tables.

The following query uses the tables emp and mgr. The name mgr is only a temporary name assigned to the emp table.

EXAMPLE: List the employees' names and salaries, and their managers' names and salaries, for employees who earn more than their managers.

```
sql> select emp .Name, emp. Salary, mgr. Name, agr .Salary
sql> from emp, emp agr
sql> where emp .Salary >= mgr. Salary and
sql>       emp. Manager = mgr.Number/
recognized query!

emp. Name|emp. Salary    |mgr. Name |mgr.Salary
Lee      |    7500.00    |Lee       |  7500.00
Colucci  |    2500.00    |Dugan     |  1650.00
Amato    |    2000.00    |Dugan     |  1650.00
```

You can join many different temporary tables, not just two. Suppose you discover that when managers are in the same location as their subordinates, employee relations are improved, and productivity increases 50%. You would naturally want to know which employees are in a location other than their managers, so you could correct the situation.

Conceptually, this query requires joining four tables: one for employees, one for employees' departments, one for managers, and one for managers' departments.

EXAMPLE: List the employees' names and locations, and their managers' names and locations, for those employees who are in locations other than their managers.

```
sql> select emp. Name, dept. Location, mgr. Name, agr_dept. Location
sql> from emp, dept, emp mgr, dept mgr_dept
sql> where emp. Manager = agr. Number
sql>       and emp.Dept_No = dept. Number
sql>       and mgr.Dept_No = mgr_dept. Number
sql>       and dept.Location ^= mgr_dept.Location/
recognized query!

emp.Name   |dept. Location|mgr. Name |mgr_dept. Location
___________|______________|__________|___________________
Smith      |San Francisco |Reilly    |Chicago
Whittaker  |New York      |Lee       |Dallas
Reilly     |Chicago       |Lee       |Dallas
Dugan      |Los Angeles   |Lee       |Dallas
```

Variable Queries

SQL evaluates standard nested queries from the inside out, passing the resulting value(s) from the innermost query to the next outer query. Each query is evaluated only once. Variable queries reverse this procedure, working from the outside in. Variable queries let you perform an inner query multiple times using values from the outer query.

Suppose you want compare salary and commission levels, and you want to know which employees earn more than 25% of the total paid to everyone else with the same job. To make sure the results are meaningful, choose the job categories with at least four employees. Take the following steps:

1. Find the job categories with at least four employees.
2. For each employee in one of these job categories, do the following:
    a. Total the salary plus commission of everyone else with the same job. Leave out the salary plus commission of the current employee.
    b. If the current employee earns more than 0.25 times this total, select this employee.

The query that determines which employees earn more than 25% of the total paid to everyone else with the same job works the same way:

1. The outer query chooses the current employee, if that person's job category has at least four employees in it.
2. The inner query uses that employee's job and number to exclude the employee from the calculation of total salary plus commission for that job category.
3. The total is multiplied by 0.25.
4. Then, if the current employee's salary exceeds that number, the employee is chosen.

This type of query is called a variable query, or a correlated query, because the inner query varies depending on a table in the outer query.

In the following example, you join the emp table to itself, and must give the emp in the outer query a temporary name, so it is apparent in the inner query which emp table you are referring to.

EXAMPLE: List the name, job, and salary plus commission for each employee who holds the same job as at least three other employees and who earns more than 25% of the total salary plus commission of the others with the same job.

```
sql> select Name, Job, Salary + Commission
sql> from emp x
sql> where Job is in select Job
sql>                  from emp
sql>                  group by Job
sql>                 having count (*) >= 4;
sql> and Salary + Commission >
sql>              select sum (Salary + Commission) * 0.25
sql>              from emp
sql>              where emp. Job = x.Job and
sql>                     emp. Number "= x. Number
sql>              group by Job/
recognized query!
```

```
Name    |Job      | Salary+Commission
________|________ |__________________
Jones   |clerk    |    900.00
Moehr   |clerk    |    950.00
Colucci |salesrep |   5500.00
```

16.2 Data Manipulation Language facilities

SQL's Data Manipulation Language facilities (DML) let you interactively insert, modify, and delete records in your database using a high-level, nonprocedural language. SQL also interfaces with the following Unify DataServer/ELS programs to improve DML performance:

- DBLOAD is the Unify DataServer/ELS database Load program (chapter 12. 6). SQL interfaces with DBLOAD so you can load a database quickly from regular ASCII files. All of the query features of the language are retained, so you can use regular query statements to insert data from existing ASCII files into other files, and select records to be modified or deleted.
- AFA is the Advanced Field Attributes program (section 8.6). SQL interfaces with AFA so SQL can validate field entries for new and existing records and set default field values for new records.
- grpmnt and empmnt are the Add or Modify Group Privileges (section 10.1) and Add or Modify Individual Privileges (section 10.3) programs. SQL recognizes access privileges set with grpmnt and empmnt for add, modify, and delete functions.

When a set of records is selected to be modified, SQL locks the table. This prevents other users from modifying the same records at the same time.

Furthermore, all the updates performed by a single DML statement are treated as a logical transaction for database recovery when transaction logging is turned on.

insert statement

The DML insert statement lets you add records to the database. This statement consists of the insert keyword followed by the into clause.

The into clause includes the name of the table to add records to, followed by constant values or values returned as the result of a query.

The insert statement can also be used to add records from an external file. For more information, see Loading Data from External Files in section 16.3.

For information on the insert statement syntax, see Formal Language Syntax in section 16.5.

You can use field value validation and default value features of Advanced Field Attributes (AFA) with the insert statement. For information on AFA, or for field validation and default value warning messages, see Advanced Field Attributes, section 8.6.

You can add specific field values to a record by listing the field names, followed by the associated field values. You must always specify primary key field names and values.

Other field values not specified or set to a predefined Advanced Field Attribute are set to the standard default values: zeros for NUMERIC, FLOAT, AMOUNT, and TIME fields, null for STRING, DATE, LDATE, TEXT, and BINARY fields.

For insert error messages refer to Messages at the end of section 16.5.

The following is an example of how to add field values into specific fields in a table:

EXAMPLE: Insert a new department, using 80 for department number, Collection for the name, and Atlanta for the location.

```
sql> insert into dept (Number, Name, Location):
sql>                    <80, 'Collection', 'Atlanta'> /
recognized update!
1 record (s) added
```

A list of constants that describes values for specific fields in a record is a literal tuple. The list of constants in a literal tuple is separated by commas and enclosed by angle brackets:

<constant1, constant2 ,...>.

You can specify a list of literal tuples in the insert statement.

If you list the fields in a literal tuple in *default field order* and all the fields are present, you can omit the field order list (in parentheses).

You can obtain the default field order of a table using the fields keyword. For example, for the emp table, you type: fields emp.

EXAMPLE: Insert three new employees as emp records.

```
sql> insert into emp:
sql>      <3000, -Owens', 80, 'clerk', 3100, 950.00, 0.00>,
sql>      <3100, 'dark', 80, 'c.p.a.', 2400, 1800.00, 0.00>,
sql>      <3200, 'Williams', 80, 'clerk', 3100, 2500.00, 0.00> /
recognized update!
3 record (s) added
```

The following example shows how you can use field value validation and default values defined by Advanced Field Attributes. The example defines only one of a total of three fields contained in the dept table. The dept table contains the fields Dept_No, Dept_Name, and Dept_Location.

SQL checks the undefined fields for Advanced Field Attribute default values, and substitutes the appropriate field values. In this example, the name and location fields have AFA defaults defined as the string "Miscellaneous".

EXAMPLE: Add a new department named Miscellaneous. Use 99 for the department number, and allow the name and location fields to default to the string "Miscellaneous".

```
sql> insert into dept (Dept_No): <99> /
recognized update!
1 record (s) added
```

You can verify the update to the dept table:

```
sql> select * from dept where Dept_No = 99 /
recognized query!
Dept_No    |Dept_Name         |Dept_Location
___________|__________________ |_____________
    99     |Miscellaneous     |Miscellaneous
```

Besides using literal tuples, an insert statement can use the results of a query statement to obtain its values.

Suppose there is a new table in the database named candidates, which contains the fields Number, Name, Dept_No, and Salary. You can select people from the emp table and put them into this new table.

EXAMPLE: Select all employees who have commissions larger than half their salaries. Make corresponding entries in the candidates table.

```
sql> insert into candidates:
sql>      select Number, Name, Dept_No, Salary
sql>      from cap sql>      where Commission > 0.5* Salary/
recognized update!
5 record (s) added
```

update statement

The DML update statement lets you modify fields in existing records. This statement consists of the update keyword followed by the set clause and the optional where clause.

The set clause lets you specify the new values of the selected fields in a record.

The where clause specifies which records to update. Otherwise, all the records for the indicated table are updated.

You can specify updates with literal values, expressions, or query statements.

You can also use the field value validation feature of Advanced Field Attributes (AFA) with the update statement. For more information on AFA, or for field validation warning messages, see Advanced Field Attributes in section 8.6. For update error messages, see Messages at the end of section 16.5.

EXAMPLE: Change Owens' salary to $1000.

```
sql> update emp
sql> set Salary = 1000
sql> where Name = 'Owens'/
recognized update!
1 record (s) updated
```

You can update a field using an expression, and you can update more than one field in a single update clause.

EXAMPLE: Move all the salesreps to department 70 and set their commissions to 35% of their salary.

```
sql> update emp
sql> set Commission = Salary * 0.35,
sql>     Dept_No = 70 sql> where Job = 'salesrep'/
recognized update!
22 record (s) updated
```

As the following example shows, you can update a field using a query statement to calculate a new value. The example also illustrates the power of the where clause. For more information on the where clause, see the where clause description in section 16.1.

EXAMPLE: For each employee in department 70, calculate the tax bill based only on salary. Store the tax bill in the Commission field.

```
sql> update emp
sql> set Commission =
sql> select Base_Tax +
sql>        ((Salary * 12) - Min_Amount) * Marginal_Rate
sql>               from emp x, taxes
sql>              where Salary*12
sql>                     between Min_Amount and Max_Amount and
sql>                     emp. Number = x. Number;
sql> where Dept_No = 70/
recognized update!
2 record (s) updated
```

delete statement

The DML delete statement lets you delete records from an existing table. This statement consists of the delete keyword followed by the optional where clause.

The delete statement indicates the table containing the records to be deleted.

The where clause specifies which records to delete. Otherwise, all the records for the indicated table are deleted.

EXAMPLE: Delete the employee named Owens.

```
sql> delete emp
sql> where Name = 'Owens '/
recognized update!
1 record(s) selected, 1 record(s) deleted
```

The following example shows how the where statement can be used to specify the records to be deleted. For more information on the where clause, see the where clause description in section 16.1.

EXAMPLE: Delete all the employees who are located in Atlanta.

```
sql> delete emp
sql> where Dept_No = select Number
sql>                   from dept
sql>                  where Location = 'Atlanta'/
recognized update!
2 record(s) selected, 2 record(s) deleted
```

16.3 Unify DataServer/ELS SQL extensions

Unify DataServer/ELS SQL has several extensions to the basic language that adapt it to interactive query development in the operating system environment:

- The most important extension for interactive querying of a database is the ability to edit the last query. To make this easier, the Unify DataServer/ELS SQL editor interface lets you edit SQL queries using your own editor without having to exit from SQL.
- Often you need to repeat queries. SQL lets you save queries in SQL script files so you can run the queries later, without re-entering them.
- You can run operating system commands from within SQL, without having to exit.
- Because of the many text processing programs available, you may sometimes want to dump data into an external file, and then load it back in the database.

This section describes how you can use these features in SQL.

Editing queries

SQL stores the last query you entered in a buffer. You can edit the query to correct mistakes, save it in a file to run later, or read in an entirely new query. To edit the current query buffer, type edit at the SQL prompt, as follows:

```
sql> edit
```

If the buffer is empty, you get an empty file to work with. The file is created with a unique name based on the process ID and is stored in the temporary work area, $UNIFYTMP.

The default editor vi is normally used. But you can specify a different editor with the EDIT environment variable. See chapter 5 for information about this and other Unify DataServer/ELS environment variables.

In the editor, you can modify the current query, type in a new query, read in a completely different query from a text file, or write the query out to a text file.

When you exit the editor, you can run the query in the buffer by entering restart at the SQL prompt, as follows:

```
sql> restart,
```

Creating and running SQL scripts

Every application has a standard set of often-used queries. Thus, the ability to store queries and run them later is important. Such stored queries are called SQL scripts.

You can create an SQL script in several ways:

1. Create a query interactively using SQL and then use edit to save it as a script.
2. Use your favorite editor outside of SQL to create the query.
3. Select "Edit SQL or RPT Command Files" from the Main Menu, enter the query, and save it.

In each case, the result is a text file containing the script.

You can include any valid SQL command except start in a stored query. Thus, a script can contain shell commands and SQL data manipulation commands. A script cannot contain the start command, because start lets you run other stored queries.

SQL has two ways to run such scripts: from SQL and from the operating system. Suppose your query, which computes the maximum salary for all employees, is stored in a file called maxsal.

From SQL

From SQL, you can run a script by entering start followed b) the file name of the script containing the query.

NOTE: If the script file name contains special characters other than the underscore, such as slashes or periods put single quote marks around it to mask the characters from SQL; for example, '../query' or 'selrec.cmd'.

To run the example script maxsal from SQL, you would use the following command:

```
sql> start maxsal
```

The query runs just as if you typed it from the terminal, except that it is not stored in the edit buffer.

From the operating system

From the shell, you can type

```
SQL maxsal
```

to run the query in maxsal. Make sure you have set the appropriate environment variables as described in chapter 5.

The results go to the standard output, while any error messages go to the standard error output. These can be piped or redirected using standard shell syntax.

The complete shell syntax for running SQL is as follows:

```
SQL [-b] script name
```

where the script name is required. The -b option tells SQL that the query results should be in a binary format. The default, without -b, is ASCII format.

Running operating system commands

You can run operating system commands from within SQL by preceding them with an exclamation point (!). For example, to list the ASCII file maxsal from within SQL, enter

```
sql> !cat maxsal
```

at the SQL prompt.

The operating system command runs, and the SQL prompt redisplays. This feature is particularly useful in stored queries where you can run a program or command without leaving SQL.

Dumping data to external files

You can send the output of an SQL query to an external file in two ways:

- from SQL
- from the operating system

From SQL

From SQL, use the keyword into to send data to an ASCII file, as shown in the following query:

```
select. * from emp into employees/
```

SQL puts the results of the query in the ASCII file called employees in the current directory.

In this example, the result is all fields from every record in the emp table. The employees file contains the heading that normally displays at the beginning of each page of output.

NOTE: The into clause is the last clause in the query. This is because SQL statements and clauses must be entered in a specific order. Thus, the statements and clauses in a longer query would be entered in the following order:

```
select...
from ...
where ...
group by...
having...
order by...
into filename /
```

From the operating system

From the shell, to send data to an ASCII file, use a stored query and redirect the standard output to a file. For example, to send the results of the previous query to a file called employees without using the into keyword, place the following query

```
select * from emp/
```

into a file called emps. Then from the operating system, use the following command to send the results to the employees file:

```
SQL emps > employees
```

Eliminating the descriptive heading

If you want to use RPT you probably don't want the descriptive heading. You would probably also want to omit the heading if you are using standard text processing filters such as *awk* or *sed*.

You can eliminate the heading by using the keyword lines. The lines setting controls the number of lines that print before a new heading prints. Setting lines to 0 suppresses the heading entirely. The default setting is 24 lines per page, or the size of a standard CRT screen. Consequently, the following command turns off the heading:

```
sql> lines 0
```

Specifying the field separator

The field separator is the character that separates database fields in an SQL listing. You can set the field separator character as follows:

```
sql> separator ' : ’
```

The default field separator is the pipe symbol (|). Any single character, including tab and nonprinting characters, is legal.

Loading data from external files

Unify DataServer/ELS SQL enables you to load data into the database from either ASCII or binary files. You can use this feature to:

1. Move data from one database to another
2. Move data from other application systems
3. Move data from one table to another

Data can be added to the database from external files (ASCII or binary) by using the DML insert statement. This statement consists of the insert keyword followed by the into and from clauses.

The into clause indicates what table to add the records to.

The from clause specifies the type and name of the file to use as input. If you are using a binary input file, the word binary follows the from clause. Otherwise, SQL assumes the input file is in ASCII format.

NOTE: Binary input files must be created using SQL only. For more information on creating binary files, see Creating and Running SQL Scripts and Dumping Data to External Files, in this chapter.

For information on the insert statement, see the explanation of the insert statement in section 16.2.

You can use field value validation and default value features of Advanced Field Attributes (AFA) with the insert statement. For more information on AFA, or for field validation and default value warning messages, see Advanced Field Attributes in section 8.6.

For insert error messages, refer to Messages in section 16.5.

Simple data loading from an ASCII file

Suppose you have an ASCII file named employees with the employee records shown in Figure 16.3.

5700	Moehr	30	clerk	6400	950.00	0.00
5800	Amato	40	salesrep	6200	2000.00	750.00
6800	Fiorella	30	clerk	5700	800.00	0.00
5900	Brown	60	engineer	1300	6000.00	0.00
6700	Colucci	40	salesrep	2200	2500.00	3000.00
7000	Simpson	40	salesrep	2300	2750.00	3500.00

Figure 16.3 Records in the ASCII File, employees

EXAMPLE: Load records from the ASCII file, employees, into the database table emp, that contains the following fields: *Number, Name, Dept_No, Position, Payroll_No, Salary, and Commission.*

```
sql> insert into emp:
sql> from employees/
recognized update!
5 record (s) added
```

Input file field order

If you do not specify the field order of an external file, SQL assumes the external file fields are in default field order. To find the *default field order* of a table, use the keyword fields. For example, for a table named emp, you would enter fields emp.

If you have an external file with fields that are not in default field order, or an external file that contains an incomplete set of fields, you must specify the actual field order in a *field order list.*

The field order list specifies the actual order of the fields in the external input file. The field order list is specified in parentheses in the insert statement, following the table name, as follows:

```
insert into emp (Name, Number, Dept_No, Salary): from empfiles /
```

If the external file contains an incomplete set of fields, the missing field values either are set to a predefined Advanced Field Attribute or are set to the standard default values. The standard defaults are zeros for NUMERIC, FLOAT, AMOUNT, and TIME fields, and null for STRING, DATE, LDATE, TEXT, and BINARY fields.

NOTE: Primary key field names and values must always be specified in the field order list and external file.

Loading data from a binary file

The following example creates a binary external file using data from a database table. The binary external file data is then loaded into another table.

NOTE: To load data from a binary external file, the binary file must have been created by SQL.

Suppose the table candidates has been added to the database. This table contains the fields Name, Number, Dept_No, and Salary. Dept_No has a default field value defined by Advanced Field Attributes of 99.

EXAMPLE: Dump a subset of the emp table to the binary file emp.file.

```
sql> select Number, Name, Salary
sql> from emp
sql> where Dept_No = 40
sql> into binary emp.file/
recognized query!
```

NOTE: To get an ASCII dump file in correct load format, you must eliminate the header at the beginning of the output, using the lines keyword. However, a binary dump file is created without a header, so the lines keyword is optional.

EXAMPLE: After the data has been dumped, load it into the new table, candidates. Allow the Dept_No field to default to 99.

```
sql>insert into candidates (Number, Name, Salary):
sql>from binary emp.file/
recognized update!
3 record(s) added
```

The insert statement explicitly specifies the input file field order using a field order list, in parentheses.

EXAMPLE: You can verify the contents of the new candidates table:

```
sql>select * from candidates/
recognized query!

Name|Number|Dept_No|Salary
_______|_________|_________|________
Amato|5800| 99 |  2000.00
Colucci|6700|99|  2500.00
Simpson|7000|99|  2750.00
```

Using text fields with SQL

In SQL TEXT fields are similar to STRING fields, with the following differences:

- You cannot use TEXT fields in a nested select statement.
- You can input a maximum of 4096 characters to a TEXT fie interactively, using insert or update.

To input a multiple-line TEXT field:

1. Start and end the TEXT string with a single quote (').
2. Insert a backslash (\) before pressing RETURN at the end of each line.

   You must include the backslash in multiple-line TEXT entries, or SQL will display an error message.

NOTE: If you insert TEXT data from another file, instead of interactively, all specified data is inserted. The 4096-character limit does not apply.

- SQL can print TEXT field data. The default number of characters that prints is the display length specified for the TEXT field in the database design.

  For example, if the TEXT field display length in the database design is 15, SQL prints the first 15 characters of the field.

  To override the display width and print the entire contents of a TEXT field, use the lines 0 command. This command prim the entire TEXT field and suppresses column headings.

  To print a specific number of characters, other than the display length or the entire field, use the text command. For example, suppose you have a file empmemos that contains employee memos. To print 50 characters from each TEXT field in the file, you would enter the following command:

```
sql> text 50 sql> select *
from empmemos /
```

NOTE: If you use the text command, it applies to all TEXT fields in the database table.

- You can search, order by, group by, and select unique TEXT fields. SQL uses as many bytes of the TEXT field as it needs to perform the operation.

Using binary fields with SQL

You can use BINARY fields with SQL with the following considerations:

- You can input a BINARY field interactively using insert or update, if you enter the information as a single string or numeric constant.
- SQL can print BINARY field data only if you use the lines 0 command. This command prints the BINARY field and suppresses column headings.
- You can order by, group by, and select unique BINARY fields. SQL considers as many bytes of the BINARY field as i needs to perform the operation.

Using SQL internal tables and environment variables

SQL has some internal tables that limit the complexity of a query. These internal tables limit expression nodes, fields, database tables, constants, selection items, sort items, and the output buffer size.

Each SQL table size can be increased or decreased to fit queries that exceed one table but barely use another. Usually, you don't have to change the SQL internal table sizes. You will have to change the internal table size if SQL displays an error message that says one of the tables is too small.

On computer hardware with limited data address space, you can reduce the SQL table sizes and make room for larger database designs. You reset a table size by setting an SQL *environment variable*.

SQL internal tables and environment variables

The following is an explanation of each internal SQL table with its environment variable. The table name is printed in boldface, and the corresponding environment variable name is in italics. In each explanation, a query refers to a complete query, from select through the terminating slash (/).

Expression Nodes{SQLNODECNT)

This table sets the number of expression nodes allowed in a query. Each expression used in a query requires one expression node. The default number of expression nodes is 100. SQL expressions are described in Query Facilities, in section 16.1.

Fields (SQLFLDCNT)

This table sets the total number of fields allowed in a query. The default number of fields per query is 100.

Constants (SQLCONCNT)

This table sets the total number of constants allowed in a query. The SQL constant table includes STRING, NUMERIC, DATE, LDATE, TIME, and AMOUNT constants. The default number of constants per query is 100. Constants and their default values are also described in Query Facilities earlier in chapter 16.

Sort Items (SQLORDCNT)

This table sets the number of sort keys allowed by sort. Queries containing the order by clause use this table. The default number of sort keys per query is 9.

NOTE: You must set this variable if the sort key capacity of sort

Tables(SQLTABCNT)

This table sets the number of database tables allowed in a query. The default number of tables per query is 10.

Selection Items (SQLSELCNT)

This table sets the number of items that can be specified in all select clauses in a query. The default number of selection items is 100.

Output Buffer (SQLPBUFSIZ)

This table sets the size of the buffer used to store a query's output items. It must be able to store the total of the output items' field lengths. The default output buffer size is 2048 bytes.

A summary of the internal SQL tables, with their corresponding environment variables is shown in Figure 16.4.

SQL TABLE	ENVIRONMENT VARIABLE	DEFAULT VALUE
Expression Nodes	SQLNODECNT	100
Fields	SQLFLDCNT	100

Figure 16.4 Internal SQL Tables and Environment Variables

SQL TABLE	ENVIRONMENT VARIABLE	DEFAULT VALUE
Constants	SQLCONCNT	100
Sort Items	SQLORDCNT	9
Database Tables	SQLTABCNT	10
Selection Items	SQLSELCNT	100
Output Buffer	SQLPBUFSIZ	2048

Figure 16.4 Internal SQL Tables and Environment Variables

SQLPMEM, SQLSMEM, AND TIMEM

Three additional environment variables control the amount of memory that SQL uses for several functions. Setting the environment variables correctly may improve the processing time for these functions.

SQLPMEM This environment variable sets the amount of memory used during an SQL sort to hold projected fields (the fields in the select clause that are not part of the order by clause).

If this variable is set too low, and the sort needs more memory for projected fields, the overflow is stored in a disk file.

This slows down sort performance. To determine how large SQLPMEM should be, use the following calculation:

total size of projected fields *
number of expected records

EXAMPLE: For example, in the following query, Job and Manager are projected fields.

```
select Dept_No, Name, Job, Manager
from emp, org
where Number = Emp_No
order by Dept_No, Name /
```

Job is 10 bytes in size and Manager is 2 bytes for a total projected fields size of 12 bytes. If you expect 300 records, your equation is:

12 * 300

for a SQLPMEM setting of

3600

SQLSMEM This environment variable sets the amount of memory used during an SQL sort to hold the sort keys. If this variable is set too low and the sort needs more room for the sort keys, the overflow is stored in a disk file. This slows down performance.

To determine how large SQLSMEM should be, use the following calculation:

(4 + sort key size) * number of expected records

EXAMPLE:For example, in the following query, Dept_No and Name are the sort keys.

```
select Dept_No, Name, Job, Manager
from emp, org
where Number = Emp_No
order by Dept_No, Name /
```

Dept_No is 2 bytes in size and Name is 10 bytes, for a total sort key size of 12 bytes. If you expect 300 records, your equation is:

(4 + 12) * 300

for a SQLSMEM setting of

4800

TIMEM This environment variable sets the maximum amount of memory used as a buffer for a temporary index. If this variable is set too low and the temporary index requires more memory, the overflow is stored in one or two disk files. If the temporary index uses less memory than specified in TIMEM, only the smaller amount is used.

There are three ways to calculate the amount of memory needed for a temporary index, because temporary indexes are used differently for each of the following SQL operations:

- sub-queries
- duplicate joins
- no-duplicate joins

After you calculate the value using each method, set TIMEM to the largest of the three values.

Determining TIMEM for Sub-queries

To determine the amount of memory needed by SQL to store temporary indexes for sub-queries, use the following calculation:

(number of expected records *
(total size of conditional fields +4)) * 2

EXAMPLE: For example, the following query contains sub-queries:

```
select Name, Job, Salary, Dept_No, Number
from emp, org
where Number = Emp_No and
      Dept_No '= 10 and
      <Job, Salary> is in select Job, Salary
                           from emp, org
                           where Number = Emp_No and
                                 Dept_No = 10/
```

The conditional fields are Job and Salary in the nested query. Job is 10 bytes in size and Salary is 4 bytes for a total conditional field size of 14 bytes. If you expect 50 records, your equation is:

(50 * (14+4))* 2

for a result of:

1800

Determining TIMEM for Duplicate Joins

To calculate the memory needed for duplicate joins, use the following formula:

```
   (number of expected records *
   total size of conditional fields +4)) * 2
   + (12) * (average number of duplicated/2))
```

EXAMPLE: For example, the following query yields duplicate joins:

```
select emp.Name, Location, mgr.Name,
       mgr_dept .Location
from emp, dept, org, emp mgr, dept mgr_dept,
       org mgr_org
where emp .Manager = mgr. Number and
       emp. Number = org.Emp_No and
       mgr. Number = mgr_org. Emp_No and
       org.Dept_No = dept. Number
       and mgr_org. Dept_No = mgr_dept. Number and
       dept. Location ^= mgr_dept. Location/
```

The conditional fields are emp.Manager, emp. Number, mgr. Number, mgr_dept. Number, and dept.Number from the where clause. The total conditional fields size is 10 bytes. If you expect 10 records and the average number of conditional duplicates that result in the same Manager is 4, your equation is:

((10 * (4 + 10)) * 2)+(12* (4/2))

for a result of:

204

Determining TIMEM for No-Duplicate Joins

To determine the memory needed for no-duplicate joins, use this calculation:

(number of expected records*
(total size of conditional fields +4)) * 2

EXAMPLE: For example, the following query yields a no-duplicate join:

```
select emp .Name, dept .Name
from emp, org, dept
where emp. Number = Emp_No and
```

```
dept. Number = Dept_No and
emp. Number > 12000 /
```

The conditional fields are Emp_No and Dept_No for a total conditional fields size of 4 bytes. If the expected number of records is 30, the equation is:

(30 * (4+4))* 2

for a result of:

480

Setting internal SQL table size

To set an SQL environment variable, use the following syntax, depending on your system:

C shell (csh):

```
setenv variable value
```

Bourne Shell:

```
variable=value
export variable
```

For example, to set the Selection Items table to 20 and the Expression Nodes table to 150 using the Bourne shell, you enter:

```
SQLNODECNT=150
SQLSELCNT=20
export SQLNODECNT SQLSELCNT
```

After you set these environment variables, you can run the SQL script again. Any changed table maximums are sent to the standard error device for SQL output.

The results of the changed environment variables would be reported as follows:

```
SQL: SQLNODECNT set to 150
SQL: SQLSELCNT set to 20
```

In general, if you are running out of memory on a query, you must decrease the size of one or more variables to increase the size of the variable (s) that needs more room.

For more information about Unify DataServer/ELS environment variables, see chapter 5.

16.4 SQL/Screen Form/Report interface

This chapter explains the following topics:

- SQL By Forms
- Registering a Screen Form with SQL
- Using an SQL Screen
- Using the SQL Report Options Screen

These topics describe the interface between screen forms, SQL queries, and the RPT report writer. The SQL/screen form/report interface lets you substitute parameter values in stored queries at run time, then send the results of the query to a report script for more sophisticated formatting.

SQL by forms (ssql)

In application systems, you usually have reports that you must run regularly. Although the format of the report remains the same, you may want to run the report for the current period, for certain customers, or for certain zip codes.

SQL By Forms gives you the ability to connect a screen form, an SQL query, and up to eight optional RPT scripts or custom programs to run such reports.

SQL By Forms consists of two parts:

- Register Screen Form with SQL
- The SQL screen driver

Register Screen Form with SQL (section 15.1 and 16.4, following), lets you associate a screen form with query and report scripts, and specify where the user can send the output of the report. The registration process creates an entry in the data dictionary referred to as an SQL screen. SQL screens can be placed on menus the same as executables (programs) and ENTER screens.

The SQL *screen driver* controls the SQL screen. The screen driver displays the screen form and lets you enter data values for each screen field. These data values will be used in the SQL script that produces the input file for the report.

After you indicate that the values are acceptable, the SQL Report Options screen lets you specify the report format, where to send the output, and whether you want to wait for the results or continue working while the report runs.

An SQL screen lets you specify values to be substituted in SQL and RPT scripts at run time. You do this by placing *parameters* in the scripts. The values you enter on the SQL screen are then substituted for the parameters when the script is run.

A parameter is a place holder in the form $n, where n is a number from 1 through the number of parameters you want the user to supply. You can place them throughout the text of a script. The script itself may be an ordinary ASCII file and registered with a screen form.

The value entered for the first screen field is substituted for $1, the second for $2, and so on.

In general, parameters in SQL scripts are used in where and having clauses to indicate what values to select. For example, to select records based on a user-supplied date range, you can use a where clause similar to the following:

```
where Hire_Date between $1 and $2
```

Or, to select departments with an average salary greater than a user-supplied value, you can use a having clause similar to the following:

```
having avg(Salary) > $1
```

You can use as many parameters in your script as there is room on the screen form. However, the number of parameters in the SQL script must equal the number of screen fields on the form. Otherwise, the screen driver cannot recognize the extra values.

For example, if you have three screen fields on a form and two parameters in the script, the screen driver can ignore the extra value. It is probable, however, that either the screen form or the script design is incorrect.

Conversely, if you only have one screen field and there is no value to fill in for the second parameter, you get a syntax error when SQL processes the script.

For these reasons, the screen driver reports an error if the number of screen fields does not match the number of parameters in the SQL script.

The screen driver substitutes parameters in the scripts just as you enter them on the screen. Thus, if you want to use a string parameter, you must enclose the parameter in single quotes.

For example, suppose you want to select records based on State_Code, a string field of length 2. The where clause is as follows:

```
where State_Code = '$1'
```

If you omit the quotes, SQL gives you a syntax error when it processes the script.

If you want the SQL output to be processed by another program, whether it is RPT, an operating system program, or a custom program, you should suppress the header lines that label the fields in the output. The way to do this is with the lines command, described in section 16.3. Add the command lines 0 to the beginning of the script to suppress these header lines.

For more information on SQL scripts, see Creating and Running SQL Scripts, also in section 16.3.

Registering a screen form with SQL (ssqlmnt)

Before SQL can drive a screen form, it must be registered with the SQL screen driver. Screens can also be registered with ENTER (section 15.1), but not with both SQL and ENTER. To determine how existing screens have been registered, run Print List of Programs (section 11.7).

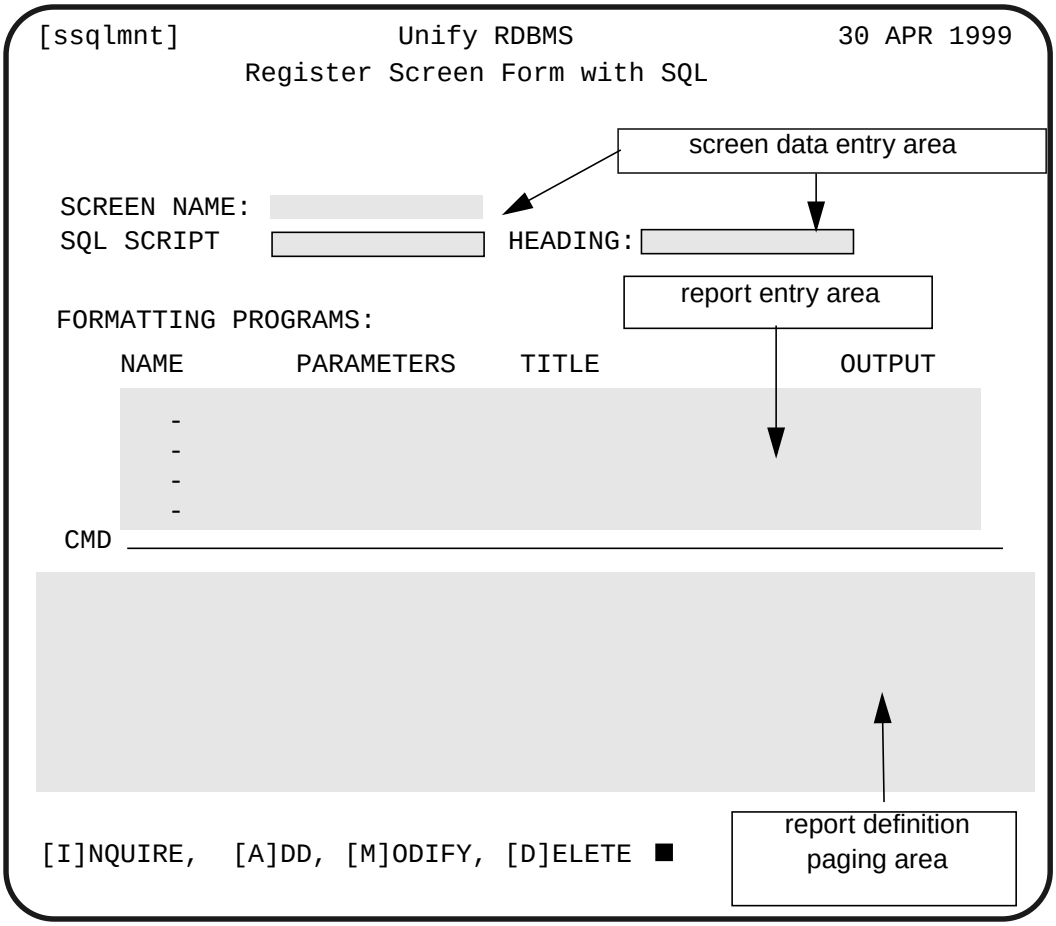
The SQL registration process tells the SQL screen driver what to display when the SQL screen is running and what SQL script to use.

This program also allows for the registration of reports. Each report is the product of a formatting program, associated parameters, title, and output options. These reports can then be run using the SQL Report Options screen (earlier in chapter 16).

After a screen form has been registered with SQL, it can also be placed on menus (section 12.2).

When you delete an SQL screen, references to that screen form such as menu listings are removed. You can obtain a list of all menus using Print Menus (section 11.2).

To register a screen form with the SQL screen driver, select "Register Screen Form with SQL" (ssqlmnt) from the Create or Modify Screen Forms menu. The following screen form displays:.



There are two entry areas on this screen: the screen form data entry area and the report entry area. There is also a report definition paging area.

The Register Screen Form with SQL screen has the following prompts and headings:

[I]NQUIRE, [A]DD, [M]ODIFY, [D]ELETE

The operation prompt. This prompt allows you to select an operation mode for the program. Figure 16.5 shows when to select each operation mode.

When you want to...	Then select...
view an existing SQL screen's registration	i (inquire)
register screen forms with SQL	a (add)
modify HEADING and FORMATTING PROGRAMS for an existing SQL screen	m (modify)
remove the SQL registration for a screen form. The deletion does not affect the original screen form. The deletion does, however, remove all references to the screen, such as menu listings.	d (delete)

Figure 16.5 Register Screen Form with SQL Operation Modes

SCREEN NAME

The name of the screen form to register. You can register one screen form with the SQL screen form driver at a time.

SQL SCRIPT The name of the file containing the SQL script to associate with the screen form.

HEADING A string of up to 34 characters that displays on the screen when the SQL screen is running and on menus where the SQL screen is a selection option.

FORMATTING PROGRAMS:

The area below this prompt is for updating and displaying the report generating processes associated with this SQL screen. The combination of the formatting program name, parameters, title, and output options is called a report.

NAME The name of the executable program that formats the output. This is the first prompt of the update area.

Enter up to eight characters for the program name. The formatting program name cannot contain embedded blanks. If you enter a name with embedded blanks, the name is truncated at the first blank. The remaining characters, including blanks, are deleted.

A maximum of eight formatting programs can be associated with each SQL screen.

For SQL Screen Registration, the formatting program can be one of the following:

1. The report processor (RPT). You must enter RPT in upper case letters. Unify DataServer/ELS executables other than RPT are not allowed.
2. A non-Unify DataServer/ELS formatting program. A formatting program that you supply is a custom program.
3. A blank. If you enter a blank, no formatting program is used, and the report lists the fields in a simple, tabular format. Therefore, a blank entry can cause output lines to wrap.

PARAMETERS

Four 14-character strings that are passed as parameters to the formatting program. The formatting program interprets the meaning of the parameters.

Enter parameters, one string per row, next to each hyphen (-). The number of parameters allowed is determined by the type of formatting program, as shown in Figure 16.6.

If the FORMATTING PROGRAM IS...	**Then the maximum number of PARAMETERS is...**
RPT	1 (the name of an RPT script)
blank	0
custom program	4

Figure 16.6 Number of Parameters per Formatting Program

If NAME is a custom program, you can enter more than one parameter in each PARAMETERS field. To do this, enter the parameters with spaces between them, providing the total length does not exceed 14 characters.

SQL starts the custom program as if you entered the command from the shell, using the following syntax:

```
SQL sql script | program p1 p2 p3 p4
```

where sql script is the name of the SQL script file registered with the screen form. program is the name of a formatting program, RPT, LST, or your custom program, p1-p4 are the four parameter strings.

TITLE — The descriptive label for the report produced when you select this option through the SQL Report Options Screen. This field and its selection number displays on the Report Options Screen.

OUTPUT — The allowed destinations for report output and the optional debug mode. Output can be routed to one or more destinations per run. The four accepted characters are shown in Figure 16.7.

When you want to...	Then enter...
enable the screen output option (DEFAULT)	S
enable the printer output option	P
enable the file output option	F
disable the debug option, nodebug (DEFAULT=enable)	N

Figure 16.7 Valid report output destinations

Lower case entries are accepted and then converted to upper case. Duplicate occurrences of a character are ignored.

Running the report in debug mode displays any error messages, the scripts used, and run time results. Having debug mode enabled lets you verify the actual script being used. Syntax errors display to help you correct program or script errors.

CMD

The area under this text prompt is the report definition paging area. The first line of each registered report displays here in the order the report appears on the SQL Report Options Screen. Individual reports are added and modified in the report entry area. However, reports are deleted directly from this area.

Control commands for a line are entered under CMD as shown in Figure 16.8.

When you want to...	Then enter...
leave the display area	q (quit)
delete the current report from this SQL screen registration	d (delete)
add a report to this SQL screen registration. You must enter a on a blank line.	a (add)
move a report to another position in the display area	the line number of the new position. The number can range from one to n, where n is the number of reports registered with this screen. The current item is moved to this line number.
modify fields associated with the current report. NOTE: If NAME is changed to blank, the parameter fields are erased. If NAME is changed to RPT, all parameter fields except the first are erased.	m (modify)

Figure 16.8 Control Commands for CMD Column

Using an SQL screen

Once you have selected the SQL screen you want to use from the Menu Handler, the screen form displays with the following prompt:

Accept entries [CTRL E], Clear field [CTRL Z], Exit [CTRL X]

This prompt indicates the default control key values for the accept entries, clear field, and exit functions. You can change the control keys by editing your termcap file as described in section 6.1.

The control characters have the following functions:

CTRL E This substitutes the field values in the SQL script and displays the Report Options Screen.

CTRL Z This sets the current selection specification to its default value.

CTRL X This exits the current SQL screen and returns you to the Menu Handler. Unlike CTRL E and CTRL Z, CTRL X can be used

whenever an ordinary character would be accepted. This gives you a quick way to exit an SQL screen.

All fields displayed on the SQL screen correspond to SQL or RPT script parameters. These fields are assigned the default display values shown in Figure 16.9.

If the field data type is...	Then the default display value is...
NUMERIC	0
FLOAT	0
AMOUNT	0.00
DATE	**/**/**
LDATE	**/**/**
TIME	00:00
STRING	(BLANK)

Figure 16.9 Standard default display formats

To specify a parameter value, enter a valid SQL expression. The expression will be substituted into the SQL script registered with this screen form.

Unify DataServer/ELS SQL string constants can contain any combination of the special characters "*", "?", and "[ ]", which are used as follows:

? The *wildcard character*. The question mark matches a single character. Thus, if you want to find all the Smiths, but don't know whether the spelling is "Smith" or "Smyth", you can use the specification Sm?th.

* The *wildcard string*. The asterisk matches a string of characters of any length, including zero-length strings, or *null strings*.

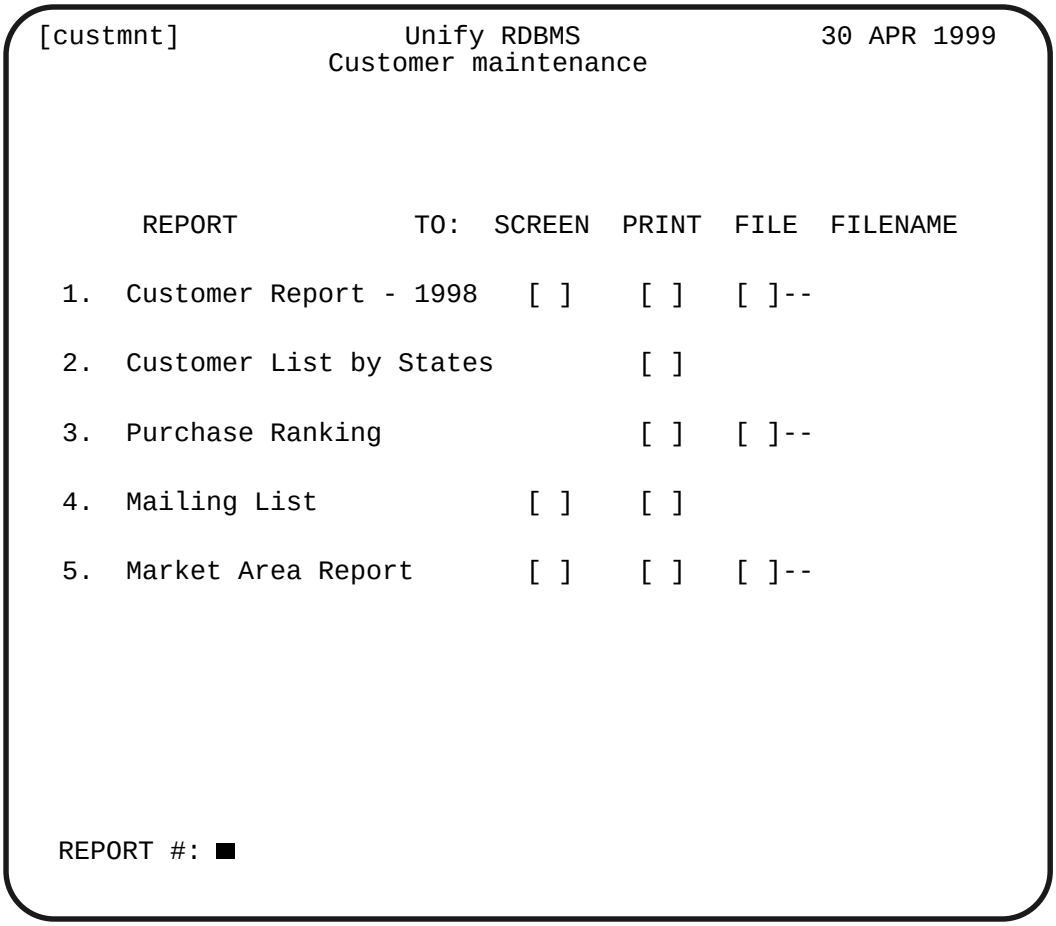
[...] The *character class*. The character class matches a single character that is a member of the class. Ranges of characters can be specified by separating two characters with a dash (-). For example, all upper case letters can be represented by the class: [ABCDEFGHIJKLMNOPQRSTUVWXYZ]

or more conveniently as [A-Z]. All upper and lower case letters can be represented as [A-Za-z].

Using the SQL report option screen

Called from SQL By Forms, the SQL Report Options Screen lets you choose a report, a combination of output destinations, and the optional debug mode for a set of retrieved records. The exact reports available and valid output destinations are set with Register Screen Form with SQL.

The following is an example of an SQL Report Options Screen:



The report Options Screen prompts are described as follows:

REPORT #: Any listed option number can be entered as a report choice. RETURN defaults to the first report item. You can also escape to the shell from this prompt using the command sh; or to a text editor using the command edit. This is useful when you want to modify formatting scripts with an editor, or you want to look at output that has been redirected to a file.

REPORT The title of the report produced when you select this option. This string derives from TITLE on the SQL Screen Registration screen.

TO: Output can be directed to the following destinations:

SCREEN The terminal screen.
PRINT The print spooler.
FILE A file with the name specified in FILENAME.

[ ] Enter an x between the brackets to select the corresponding output destination for the current report. Any combination of available options is valid.

-- When the FILE output destination option is chosen, the output file name must follow this prompt. If a file with that name exists, the following message displays:

File already exists, destroy it?

Answer Y to write over the existing file, or answer N tc enter a new file name.

When you choose a valid report item with REPORT #:, the requested item line is highlighted and the cursor moves to the firs allowed output destination prompt. After you choose the appropriate destinations, return to the first output prompt with CTRL U or RETURN. Enter CTRL U to continue to the process mode prompt.

If the chosen report has only one valid output destination, that option is automatically requested.

The *process mode* prompt then displays on the REPORT #: line:

Proceed in [F]oreground, [B]ackground, [D]ebug or [C]ancel?

This prompt lets you choose the process mode for the report.

The [D]ebug option does not display if nodebug mode (N) has been set with SQL Screen Registration. The [B]ackground option is not allowed if SCREEN is chosen as an output destination. The possible selections are as follows:

f Process the report in foreground mode (while you wait).

b Process the report in background mode (while you continue working on something else).

d Process the report in foreground mode and display the scripts, run time results, and error messages if applicable. Debug is useful because it displays the scripts used with the argument substitution.

c Cancel the report request and return to the REPORT #: prompt.

The Report Options Screen may also display any of the following messages at run time:

<EMPTY>

Explanation: The output holding file has no data.

Complete. Please enter RETURN to continue. ->->

Explanation: This part of the process is finished.

Display next page? [RETURN] continues, In] terminates

Explanation: The output from the current process does not fit on one screen form. RETURN displays the next screen of data, while n ends the display and the process continues.

Unable to create output file, please reenter filename

Explanation: During the process, the given filename (FILENAME) could not be created.

16.5 SQL reference

This subsection contains reference information for those who know how to use SQL and just need quick information. It explains the reserved keywords used by SQL.

This subsection also contains diagrams of the formal language syntax of the complete Unify DataServer/ELS SQL language. These diagrams can show you how to construct any kind of SQL statement. At the end of this subsection is a summary of the error messages and possible actions to correct problems.

Keyword summary

The keywords listed in Figure 16.10 have specific meaning for SQL. Therefore, you cannot use keywords for table or field names. Queries using these keywords as table or field names result in syntax errors.

SQL KEYWORDS

and	asc	avg	between	binary
by	count	delete	desc	edit
end	fields	from	group	having
help	in	insert	into	is
lines	max	min	not	or
order	restart	select	separator	set
start	sum	tables	text	unique
unlock	update	where	write	

Figure 16.10 SQL Keywords

Formal language syntax

Unify DataServer/ELS SQL consists of four basic parts:

- The query language T
- The data manipulation language
- The command language

- SQL help

The *query language* is the largest part. It lets you form questions about the information in your database by asking "what" and letting SQL determine "how" to get the information.

The *data manipulation language*, or DML, lets you modify the database by inserting, updating, and deleting records.

The *command language* has different capabilities that make the interactive development and execution of query and DML statements easier. These capabilities include editing the current SQL statement, starting and restarting queries from script files, unlocking fields protected by field level security, and executing operating system commands.

SQL *help* lets you quickly review how a particular command is used, and lets you list the valid table and field names in the current database.

Query statements and clauses

An SQL query consists of one or more query blocks, and is terminated by a slash (/). A query block is described as follows:

1. It starts with a select clause that specifies the fields or expressions to be retrieved.
2. It continues with a from clause that specifies the tables containing the fields to be retrieved.
3. It can include one or more optional clauses.

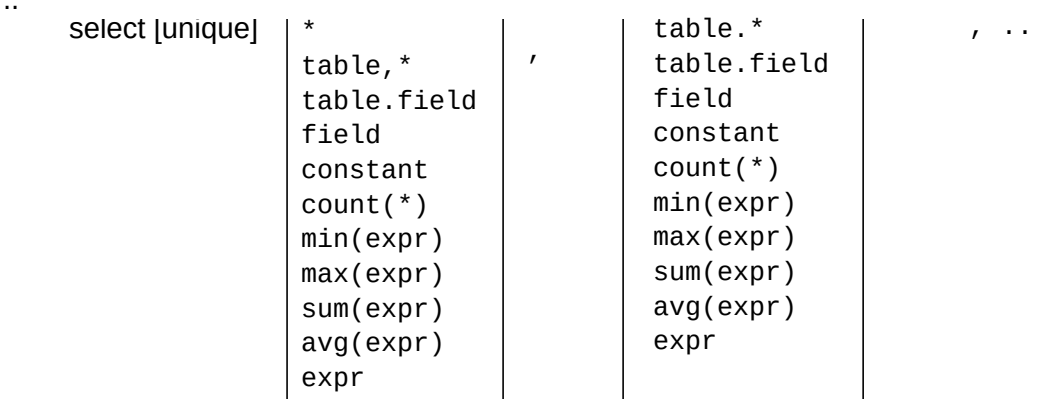
Although optional clauses may be omitted in a specific query, the statements and clauses in an SQL query block must be entered in the following order:

```
select...
from ...
[ where ... ]
[ group by ... ]
[ having ... ]
[ order by ... ]
[ into ... ] /
```

The dots represent the continuation of the clause. This subsection describes the syntax and options for each of the following query statements and clauses:

- select Statement
- from Clause
- where Clause
- group by Clause
- having Clause
- order by Clause
- into Clause

select SATEMENT



The select statement lets you specify what fields or expressions to retrieve from your database. Every query must start with a select clause.

Arguments

unique The unique keyword is used to eliminate duplicate records from the query result. If every column in two or more records matches, only one row is displayed in the result. This keyword causes the query output to be sorted.

*	The asterisk is a wildcard used to select all fields in all the tables named in the from clause. If you use an asterisk, no other fields or expressions can be included in the select clause.
table.*	The name of a table in the database, followed by a period and an asterisk. The result is to select all the fields from the indicated table.
table.field	The name of a field, preceded by the name of the table that field is in. If the table has been assigned a label, or temporary name, in the from clause, then the label can be used in place of the table name. You must precede the field name by the table name this way when you use a field name that occurs in two or more tables listed in the from clause. Without this qualification, it is not clear which table the field is in.
field	The name of a field in one of the tables listed in the from clause.
constant	A constant value of type NUMERIC, FLOAT, AMOUNT, DATE, LDATE, TIME, or STRING. The type of constant depends on the type of field or expression compared with the constant. Enclose STRING constants in single quotes (') to distinguish them from field names.
count(*) min(expr) max(expr) avg(expr) sum(expr)	These are the built-in SQL aggregate functions. When you use an aggregate function, it implies a group by should be performed. If you haven't specified the group explicitly, the entire query result is treated as a single group. Using an aggregate function also implies that you have selected only fields or expressions whose values are common to all records in the group. The count(*) function counts the number of records in each group that matched the query selection criteria.

The min, max, and avg functions calculate the minimum, maximum, and average values of the indicated expression for the records in each group that matched the query selection criteria.

The sum function totals the indicated expression for the records in each group that matched the query selection criteria.

Aggregate functions can be applied to fields of type NUMERIC, AMOUNT, FLOAT, DATE, LDATE, and TIME, although the results on DATE, LDATE, and TIME fields can be meaningless. Also, aggregate functions can be nested two levels to achieve such expressions as avg(count(*)) or min(avg(salary)).

expr — An arithmetic expression, containing fields, constants, and aggregate functions, connected by the operators +, -, *, and /. Parentheses can be used to change the order in which the operations are performed.

from CLAUSE

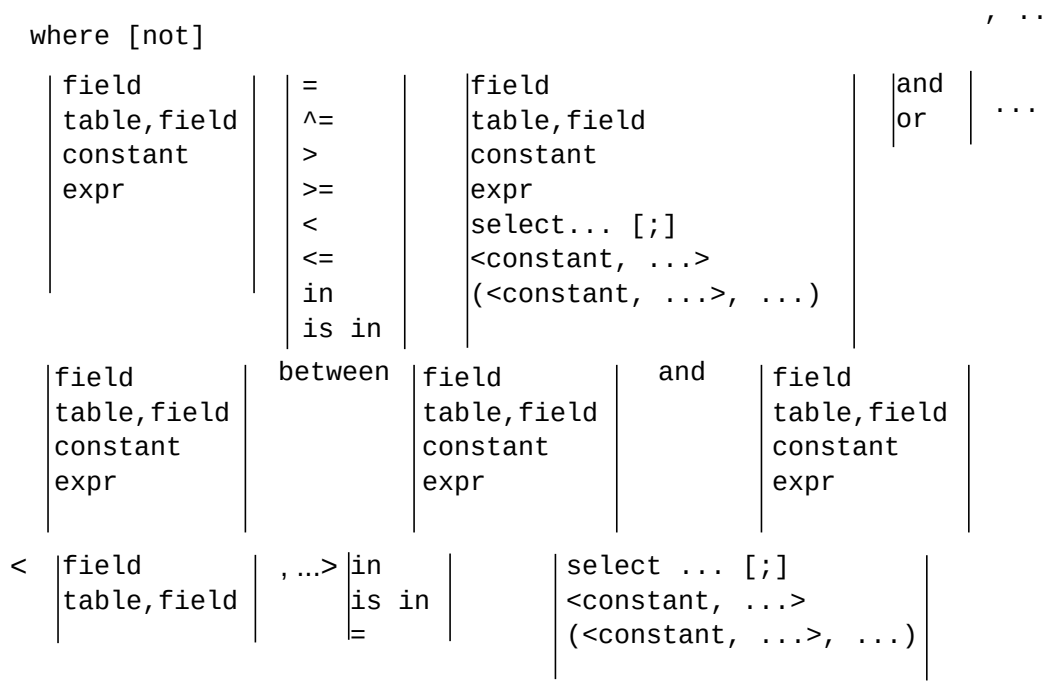
from | table [label] \, ...

The from clause lets you specify the tables to be used in the query. You can list any number of tables, in any order. SQL determines the best way to retrieve the data.

Arguments

table — The name of a table in the current database.

label — A label that identifies the table later in the query. Labels are temporary names used in self-join and variable queries. For more information, see General Join, in section 16.1.



The where clause lets you specify selection criteria to select reject records in the query result.

The syntax shows that a where clause can consist of one or more Boolean expressions connected with and or or. Each Boolean expression can have one of the following three forms:

1. Using the comparison operators.
2. Using the between-and operators.
3. Using the set equality operators.

If a where clause contains multiple Boolean expressions connecte with either and or or, square brackets can be used to indicate th expression to evaluate first. Otherwise, the default order is and expressions, then or expressions, from left to right.

Arguments

field — The name of a field in one of the tables listed in the from clause.

table.field — The name of a field, preceded by the name of the table that contains the field. If the table has been assigned a label, or temporary name, in the from clause, then the label can be used in place of the table name.

You must precede a field name by a table name this way when you use a field name contained in two or more tables listed in the from clause. Without this qualification, it is not clear which table the field is from.

constant — A constant value of type NUMERIC, FLOAT, AMOUNT, LDATE, TIME, or STRING. The type of constant depends on the type of field or expression compared with the constant.

STRING constants are enclosed in single quotes (') to distinguish them from field names.

expr — An arithmetic expression, composed of fields and constants, connected by the operators +, -, *, and /. Parentheses can be used to change the order in which the operations are performed.

select... — This indicates that a query block can be used in this part of the clause.

<constant, ...> — This indicates that a constant tuple or literal tuple can be used instead of a single value when comparing for equality with the left side of a Boolean expression. A literal tuple is a list of constants separated by commas and enclosed in angle brackets.

This kind of expression can be used as follows:

field is in <c1, c2, c3>

This is equivalent to the more cumbersome expression:

field = c1 or field = c2 or field = c3

(<constant, ...>,...) — This indicates that a list of literal tuples can be used when comparing

for equality with the left side of a Boolean expression. Each tuple list is constructed with commas separating the tuples, and with parentheses enclosing the list, as follows:

<f1, f2> is in (<c1, c2>, <c3, c4>)

This is equivalent to the more cumbersome expression:

[f1 = cl and f2 = c2] or [1] = c3 and f2 = c4]

=, ^=, >, >=, <, <=
The comparison operators. In order, they are equal, not equal, greater than, greater than or equal to, less than, and less than or equal to.

in
is in
These two operators are equivalent to the equal sign. They are provided so queries involving literal tuples read naturally. You can use either these operators or the equal sign (=).

between - and
This is a single operator that makes ranges easier to specify. The normal way to specify a range in a query language is as follows:

field <= c1 and field >= c2

Using this operator, the expression reads:

field between c1 and c2

This is a more natural way to specify ranges.

and
The conjunction operator. Two simple Boolean expressions can be connected with and to form a compound Boolean expression. The compound expression is true if both simple expressions are true. It is false if either simple expression is false.

or
The disjunction operator. Two simple Boolean expressions can be connected with or to form a compound Boolean expression. The

compound expression is true if either simple expression is true. It is false if both simple expressions are false.

group by CLAUSE

group by | field , ... | table,field |

The group by clause lets you sort records into groups so you can apply an aggregate function to each group. Using a group by without an aggregate function is meaningless.

The group by clause implies that each field or expression in the select clause has the same value for each record in that group. If you list fields or expressions that don't have the same value for every record in the group, the query result is not very meaningful.

For example, if you select employee name and group by department number, the name that displays isn't necessarily the only employee name in the department.

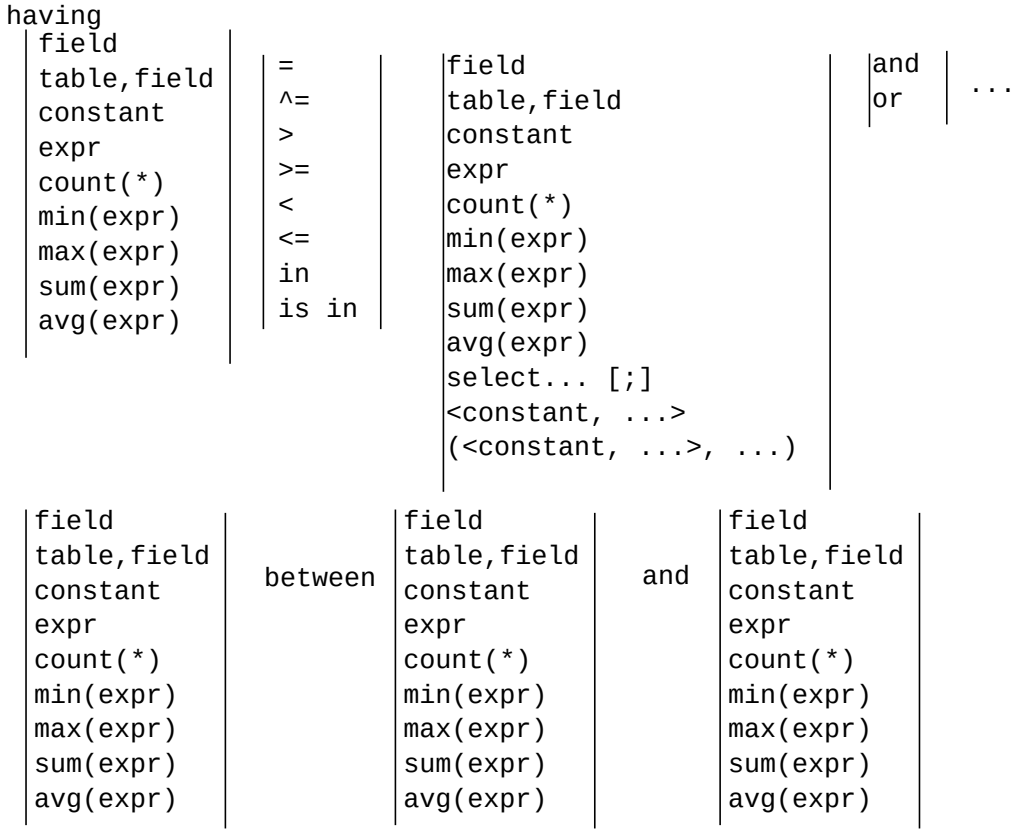
Arguments

field — The name of a field in one of the tables listed in the from clause.

table.field — The name of a field, preceded by the name of the table that contains the field. If the table has been assigned a label, or temporary name, in the from clause, the label can be used in place of the table name.

You must qualify a field in this way when you use a field name that occurs in two or more different tables listed in the from clause. Without this qualification, it is not clear which table the field is from.

having CLAUSE..



The having clause lets you select some groups formed by the group by clause and reject others, based on the value of an aggregate function. Therefore, the having clause must contain at least one aggregate function, and it must be preceded by a group by clause.

A having clause is composed of one or more Boolean expressions connected by *and* or *or*. Each Boolean expression can be in one of two forms:

1. Using the comparison operators.
2. Using the between-and operator.

If a having clause contains multiple Boolean expressions connected with either and or or, square brackets can be used to indicate the expression to evaluate first. Otherwise, the default order is and expressions, then or expressions, from left to right.

Arguments

field	The name of a field in one of the tables listed in the from clause.
table.field	The name of a field, preceded by the name of the table that contains the field. If the table has been assigned a label, or temporary name, in the from clause, then the label can be used in place of the table name. You must qualify a field this way when you use a field name that is duplicated in two or more tables listed in the from clause. For example, both the emp and the dept tables contain a field called Number. If you do not specify the table, SQL cannot determine which Number field you want.
constant	A constant value of type NUMERIC, FLOAT, AMOUNT, LDATE, TIME, or STRING. The type of constant depends on the type of the field or expression compared with the constant. STRING constants are enclosed in single quotes (') to distinguish them from field names.
expr	An arithmetic expression, composed of fields, constants, and aggregate functions connected by the operators +, -, *, and /. Parentheses can be used to change the order in which the operations are performed.
NOTE:	The subtraction operator must be surrounded by spaces to distinguish it from the minus sign.
count(*) min(expr) max(expr) avg(expr) sum(expr)	These are the built-in SQL aggregate functions. When you use an aggregate function, it implies a group by should be performed. With a having clause, you must specify the group explicitly. Using an aggregate function also implies that you have selected only fields or expressions whose values are common to all records in the group.

The count(*) function counts the number of records in each group that match the query selection criteria.

The min, max and avg functions calculate the respective minimum, maximum, and average values, of the indicated expression for the records in each group that match the query selection criteria.

The sum function totals the indicated expression for the records in each group that match the query selection criteria.

Aggregate functions may be applied to fields of type NUMERIC, AMOUNT, FLOAT, DATE, LDATE, and TIME, although the results on DATE, LDATE, and TIME fields can be meaningless. Also, aggregate functions can be nested to achieve such expressions as avg(count(*)), and min(avg(salary)).

select ... This indicates that a query block can be used in the having clause. For more information on query blocks, refer to the beginning of Query Statements and Clauses earlier in this chapter.

<constant, ...> This indicates that a constant tuple or literal tuple can be used instead of a single value to compare for equality with the left side of a Boolean expression. A literal tuple is a list of constants, separated by commas, and enclosed in angle brackets.

Literal tuples can be used as follows:

```
field is in c1, c2, c3
```

This is equivalent to the more cumbersome expression:

```
field = c1 or filed = c2 or field = c3
```

(<constant, ...>,...)
This indicates that a list of literal tuples can be used to compare for equality with the left side of a Boolean expression. A tuple list is constructed with commas separating the tuples and with parentheses enclosing the list of tuples.

Literal tuple lists can be used as follows:

<f1, f2> is in (<c1, c2>, <c3, c4>)

This is equivalent to the more cumbersome expression:

[f1 = c1 and f2 = c2] or [f1 = c3 and f2 = c4]

= ^= > >= These are the comparison operators. In order, they are equal, not

< <= equal, greater than, greater than or equal to, less than, and less than or equal to.

in These two operators are equivalent to the equal sign. They are

is in provided so queries involving literal tuples read naturally. You can use the equal sign for exactly the same effect.

between - and
This is a single operator that makes ranges easier to specify. The usual way of specifying a range in a query language is as follows:

field <= c1 and field >= c2

Using the between-and operator, the expression reads:

field between c1 and c2

This is a more natural way to specify ranges.

and The conjunction operator. Two simple Boolean expressions can be connected with and to form a compound Boolean expression. The compound expression is true if both simple expressions are true. It is false if either simple expression is false.

or The disjunction operator. Two simple Boolean expressions can be connected with or to form a compound Boolean expression. The compound expression is true if either simple expression is true. It is false if both simple expressions are false.

order by CLAUSE

```
..
 order by   | field        [ |  asc   | ]   |, ...
            | table,field    |  desc  |     |
```

The order by clause lets you sort the output of a query. You can specify ascending or descending sorts on each field. If you omit the specification, the default sort order is ascending.

Arguments

field — The name of a field in one of the tables listed in the from clause.

table.field — The name of a field, preceded by the name of the table that contains the field. If the table has been assigned a label, or temporary name, in the from clause, that label can be used in place of the table name.

You must qualify a field this way when you use a field name that occurs in two or more tables listed in the from clause. Without this qualification, it is not clear which table the field is from.

asc

desc — These keywords indicate the order in which the preceding field is to be sorted. If you want ascending order, use asc, and if you want descending order, use desc.

into CLAUSE

into [binary] filename

The into clause lets you send the results of the query to an ASCII or a binary file, instead of the CRT screen. This is useful if you want to use the data in another program, or load it back in another database.

The binary file option can increase performance by not requiring that data be converted to ASCII and then reconverted to binary during a data dump and reload.

Arguments

filename — The name of the query output file. If you use the binary option, this file will contain binary data.

An error occurs if a file by the specified name exists. If the file name contains such special characters as a dot or slash; e.g., ,./xxx.oiit or data. txt, you must enclose the file name in single quotes (').

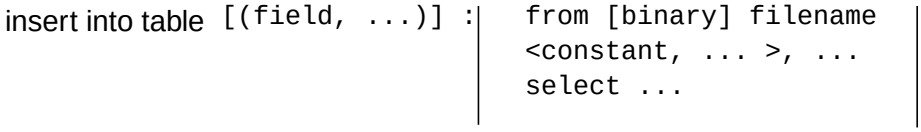
DML statements and clauses

An SQL data manipulation clause consists of an introductory keyword statement that indicates the update to be performed, whether insert, update, or delete, followed by the clauses that specify how to perform the update. The following lists the DML statements and clauses:

- insert statement
- update statement
- delete statement
- set clause

As with query language statements, DML statements must be ended with a slash (/). Much of the power of query statements can be used in data manipulation, which lets you operate on large sets of records with a single statement. The following describes the syntax of each data manipulation statement.

insert Statement ..



The insert into clause lets you add records to a database. You can use an ASCII file, binary file, literal tuples, or the results of a query to obtain the data values for the new records.

Arguments

table — The name of a table in the current database.

(field, ...) — The optional field specification list. It consists of a comma-separated list of field names, enclosed in parentheses. The list defines which fields are to be loaded and the order they occur in the input stream, whether that stream comes from an ASCII file, literal tuples, or from a query statement.

If you omit the list, the input stream is assumed to contain all fields in the same order as would be returned from the following query:

```
select * from table l
```

This is the default field order for a table.

filename — The name of a file that contains data to be loaded into the database. The keyword from must be used in this clause.

The file can be either an ASCII or a binary file. To specify a binary file, you must use the binary keyword option. Using a binary file can increase performance, because the data does not have to be converted from ASCII.

If the file name contains special characters such as a dot or slash, then it must be enclosed in single quotes; e.g., '../xxx.out'.

The order of fields in the file must either be the same as the field specification list, or the same as the default field order for the table being loaded.

<constant, ...>,... — This indicates that a constant tuple or literal tuple can be used to specify the values for the fields in the field specification list. The order of fields in the literal tuples must either be the same as the field specification list, or the same as the default field order for the table being loaded.

select ... This indicates that a query block can be used in this part of the statement. The order of the fields returned from the query must either be the same as the field specification list, or the same as the default field order for the table being loaded. For more information about query blocks, refer to Query Statements and Clauses earlier in this chapter.

update statement

update table set ...[where...]/

The update statement modifies the specified fields in all records that meet the selection criteria in the where clause. If the where clause is omitted, every record is updated.

Arguments

table The name of a table in the current database.

set A set clause is inserted here. For more details, refer to the set description following this update description.

where An optional where clause can be placed here to specify the records to be updated. For more information, refer to the where clause description in Query Statements and Clauses, earlier in this chapter.

delete statement

delete table [where ...] /

The delete statement lets you delete specific records in a table based on the results of a where clause. If you omit the where clause, every record in the table is deleted.

Arguments

table The name of a table in the current database.

where An optional where clause can be placed here to specify the records to be deleted. Refer to the previous where clause description, earlier in this chapter, for more information.

set CLAUSE ..

```
set   | field = expr        |, ...
      | field = select ...  |
```

The set clause lets you indicate how to update the fields in a table. You can set a field to an expression or to the result of a query statement.

Arguments

field — The name of a field in the table that is being updated.

expr — An arithmetic expression, composed of fields and constants connected by the operators +, -, *, and /. The fields must be in the table being updated. Parentheses can be used to change the order in which the operations are performed.

select ... — This indicates that a query block can be used in this part of the clause. For more information about query blocks, refer to Query Statements and Clauses, earlier in this chapter.

Non-Query commands

Unify DataServer/ELS SQL includes the following commands that augment the language:

- edit
- restart
- start
- unlock
- lines
- text
- separator
- ! (run operating system command)
- end

The edit, restart, and start commands enable you to change, then rerun the current query, or run queries stored in ASCII files. The unlock command enables you to access fields protected by passwords. The lines and separator commands let you set the number of lines per page for the column headings and change the standard field separator. The command (!) lets you execute operating system commands. And when you finish using SQL, you can exit with the end command. The following explains each of these commands.

edit COMMAND

edit — The edit command lets you modify the current SQL query. The query is stored in a temporary file in the directory specified in UNIFYTMP. You can correct syntax mistakes or modify the query to produce a different result.

The default editor is vi. If you prefer a different editor, set the EDIT environment variable to specify the editor of your choice. For more information on environment variables, see chapter 5.

restart COMMAND

restart — The restart command lets you run the current SQL query stored in the temporary file. When you enter a new query, it becomes the current query, replacing the old one in the temporary file. You usually use this command after you edit the current statement.

start COMMAND

start filename — The start command runs the SQL query stored in the named operating system file. Running a query this way does not cause it to become the current query. The current query remains the last one you entered.

Arguments

filename — The name of an ASCII file that contains an SQL query. The query runs and the results display on the screen. If the file name contains special characters such as a dot or slash, then it must be enclosed in single quotes to mask the characters from SQL; e.g., '../xxx.out'.

unlock table, field, password [write] [, table, field, password [write] ...] /

The unlock command lets you access database fields that have been protected by Security Maintenance (section 10.4). This command must be ended with a slash (/) just like a query.

Arguments

table.field — The name of the database field you want to unlock, the field name must be preceded by the name of the table that contains the field.

password — If the write keyword is not specified, this is the read password entered in Field Security Maintenance. If write is specified, this is the write password.

lines COMMAND

lines number — The lines command changes the line number where a new heading is printed. The default is set to 24 lines, because that is the size of most CRT screens. Thus, a new header is printed every 24 lines.

Arguments

number — A number between 0 and 32767 that indicates the line count where a new heading prints. If the number is 0, no heading is printed. Otherwise, a heading is printed at the start, and after every number lines of output.

You can use lines 0 to reduce the output to a minimum of characters, for instance for input to a pipeline, or to distinguish a null field from a field that is filled with blanks. For any STRING field that is null or blank, lines 0 prevents padding the field with spaces and instead displays the field in a condensed format. For example, if your separator character is a vertical bar (|), a null field is displayed as ||.

NOTE: To display an entire TEXT field, not merely the field display length, you must use lines 0.

text COMMAND

text number The text command specifies the number of TEXT field characters that display. The default is the display length specified for the TEXT field in the database design, unless the lines 0 command is entered. If you enter the lines 0 command, the entire TEXT field prints.

Arguments

number A number that indicates how many characters to display for each text field.

separator COMMAND

separator ' character'

The separator command lets you change the field separator that SQL uses to indicate where one field ends and the next starts. The default is the vertical bar (|).

Arguments

character A single character that is the new field separator. This can be any character, including nonprinting control characters.

If you want to load in an ASCII file using the insert statement, the ASCII file must use the same field separator throughout. Also, SQL's field separator must match the one in the file you want to load.

Run operating system commands

! operating-system-command

The ! command lets you run operating system commands. Any command preceded by an exclamation point is passed to the operating system. When the command has finished running, SQL regains control.

end COMMAND

end — The end command lets you exit SQL and return to either the Menu Handler or the operating system (depending on where you started from). Enter end at the SQL prompt.

HELP commands

An SQL help command consists of one or two keywords, the second indicating the subject of the help information. Help information is a short reminder on the syntax of any particular statement or command.

SQL has the following three help commands:

- help
- tables
- fields

Once you are familiar with SQL, you can use the online help as a quick reference, instead of returning to the manual. The following describes how to use SQL help.

help COMMAND

help [keyword]

The help command displays help information on the use of SQL. If you enter help, you see a general message about how to use the help features. If you enter help keyword, you see help information about that keyword.

Arguments

keyword — A keyword used by SQL. If there is help documentation about that keyword, it displays. Keywords are listed in Keyword Summary at the beginning of the SQL Reference chapter.

tables COMMAND

tables — The tables command lists the names of the tables you can use in constructing SQL statements. This command has no options.

fields COMMAND

fields table — The fields command lists the fields for the indicated table, with their data types and lengths. Fields are listed in the order specified by field_number in the field table of unify.db.

Arguments

table — The name of a table in the current database. If you don't know the table names, use the tables command.

SQL errors

SQL has two classes of errors: syntax errors and semantic errors.

Syntax errors are errors in the structure of the statement, such as misplacing or misspelling a keyword. Syntax errors prevent SQL from recognizing your query.

Semantic errors are errors in the meaning of the statement, such as selecting a unique field such as employee name and grouping by department number. The statement is valid grammatically, but the results aren't very meaningful.

Thus, a semantic error can occur when you use aggregate functions and you list a field that is not a property of the group used to compute the aggregate. In this case, SQL does not report an error. But the value (s) listed for the indicated field (s) has nothing to do with the aggregate function value returned.

For example, suppose you want to know the employees who earn a salary equal to the average salary for all employees. To answer this question, you can use the following query :

```
select name, avg (salary) from emp/
```

This query does not result in an error. It does produce a name and the average salary for all employees, but the name returned is merely the last one read. This employee may or may not earn the average salary. In fact, other employees who are not listed may earn the average salary.

To list the employees who earn a salary equal to the average salary of all employees, you must use a nested query similar to the following one:

```
select name, salary from emp
where salary =
   select avg(salary)
   from emp/
```

This query asks two questions:

1. The inner query asks "What is the average salary for all employees?"
2. The outer query asks "What employees make that salary?" For more information on nested queries, see Query Facilities at the beginning of section 16.1.

Part VI: Report generators

Part VI, Report Generators, describes the two ways you can produce reports in Unify DataServer/ELS: using RPT and using LST.

Chapter 17, RPT-Report processor, describes RPT. RPT is a report generator with a powerful, non-procedural language you can use to format customized reports.

Chapter 18, LST-Listing processor, describes LST. LST is a simpler tool you can use to produce quick reports, without writing an RPT script.

Chapter 17: RPT–Report processor

This chapter describes the Unify DataServer/ELS Report processor, RPT. The first part of this chapter introduces the concepts of report writing and briefly describes how RPT fits in with the rest of Unify DataServer/ELS . The second part contains tutorial type examples that illustrate the features of RPT. The remainder of the chapter is reference material describing RPT syntax, keywords, and error messages.

17.1 Report writing concepts

The Unify DataServer/ELS report processor, RPT, includes a nonprocedural language that lets you quickly and easily specify the format of multiple-level, tabular reports. The data you want to format can come from an SQL or Query By Forms query, from a user program, or from an ASCII file.

You can control the placement and format of fields, headings, footings, column titles, and pagination. Thus, you can design the appearance of the report. This chapter describes how to create an RPT report and defines the terms you use.

Files used by RPT

You need two files to produce a report with RPT: *an input file* and a *report scrip*t file.

The *input file* contains the data you want to format and summarize in a report. The input file is an ordinary ASCII file composed of identical lines. Each line has a number of data fields and ends with a new-line character. You can use such tools as SQL, Query by Forms, a user program, or a text editor to create input files.

The *report script* contains report processor commands. The commands describe the format of the lines in the input file and control the appearance of the finished report. RPT reads the input file and uses the commands in the report script to produce a

report. The diagram in Figure 17.1 illustrates the relationships among input files, report scripts, and RPT.

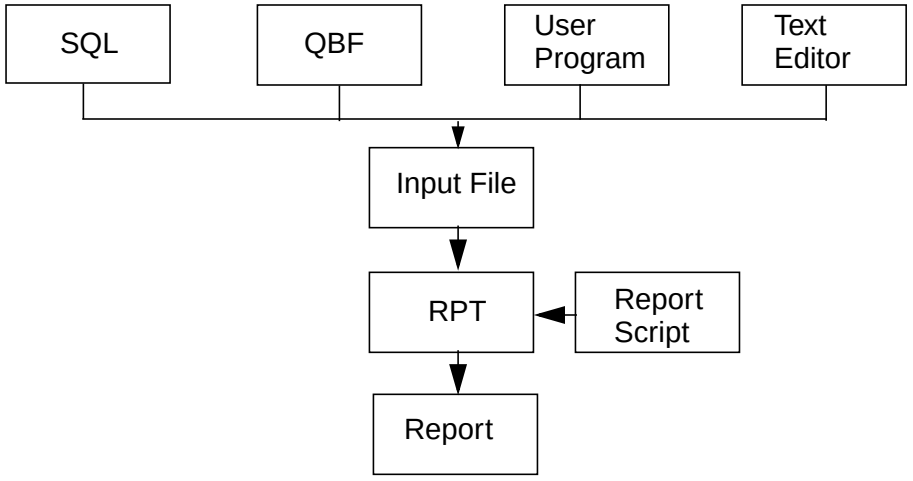


Figure 17.1 Relationships between rpt and other programs

For more information on using RPT with SQL, see SQL Screen Form/Report Interface, section 16.4, and Using RPT with other Unify DataServer/ELS tools, section 17.10. For information on combining RPT with ENTER or with user programs and shell scripts, see Register Screen Form with ENTER, section 15.1, or Using ENTER screens, section 15.2, and Using RPT with Other Unify DataServer/ELS tools, section 17.10.

Example report

To understand how the report writer operates, you need to understand the structure of an RPT report and the terms that describe the parts of a report.

Suppose you have a small database that contains three tables: employees, departments, and taxes. This is the same design used in the SQL description in chapter 16. And suppose you want a report that lists all departments, each employee's name, job, salary, and commission, with a total for each department, and a grand total for the company.

Figure 17.2 contains an example of such a report.

```
                Department Listing                                  <-- before report
10 Administration                                                   <-- before dept_no
Name            Job                  Salary          Commission
--------------- -------------------- --------------- ---------- <-- detail line
Bleriot         programmer                 1100.00       0.00
Jones           clerk                       900.00       0.00
Lee,            president                  7500.00       0.00
Scharf          clerk                       800.00       0.00
                Total                     10300.00       0.00 <-- after dept_no
20 Eastern Sales
Name            Job                  Salary          Commission
--------------- -------------------- --------------- ----------
Klein           salesrep                   1500.00       0.00
O'Neil          salesrep                   1500.00     150.00
Whittaker       salesrep                   2500.00     500.00
                Total                      5500.00     650.00
70 Finance

Name            Job                  Salary          Commission
--------------- -------------------- --------------- ----------
Fiorella        clerk                       800.00       0.00
Moehr           clerk                       950.00       0.00

                Total                      1750.00       0.00
                Grand Total               36600.00    8800.00 <-- after report
```

Figure 17.2 Example report

The *input file* consists of a single line for each employee in the company**.** Each line contains fields for the department number, department name, employee name, job title, salary, and commission. The report script that specifies the format of the output is described a step at a time in Basic Report Examples in this chapter.

Notice that the report is *sorted* by department number and employee name. Each field you sort by is called a sort key. An RPT sort key can be a simple database field or a complex expression composed of fields, constants, variables, arithmetic operators, and functions.

Sorting has two purposes:

1. To arrange items with different sort keys in ascending or descending order, e.g., employees with different names.
2. To group items with the same sort key, e.g., all employees in department 10.

When the value of a sort key changes, a *control break* occurs. The control break separates groups of items with different sort keys. For example, when the employee department number changes from department 10 to department 20, a control break occurs.

RPT lets you specify printing and processing actions both before and after the control breaks that separate groups of items. For example, suppose the control break from department 10 to department 20 occurs. After the department 10 group, you want to print the totals for department 10. Before the department 20 group, you want to print the department number, name, and column headings for the items that follow.

A control break also occurs in the transition from "no data" to "data," as for the first department in the report.

There are five other kinds of control breaks: detail, before and after, and header and footer. The detail control break occurs every time a new line is read from the input file. This is the lowest level break. Use it to print a line or add a number to a total. The before report and after report control breaks always occur at the beginning and end of the report. Use the before report break to print report headings or titles. Use the after report break to print a report grand total.

The header and footer control breaks occur at the top and bottom of each report page. Use the header control break to print a running title, date, time, or page number. Use the footer control break to print page numbers or a brief note.

The report script sections that describe control break commands are processed in the following sequence:

```
before report
before sort key n
        "
        "
        "
before sort key 1
```

```
detail
after sort key 1
        "
        "
        "
after sort key n
after report
```

The header and footer control breaks are not included in this hierarchy, because they occur before and after a new page. This depends on the page size. For example, the same report using the same data can have its header and footer control breaks in different places if it prints on two sizes of paper.

Therefore, header and footer control breaks float, occurring at a transition to a new page.

Section 17.2 explains these terms by presenting example reports and their report scripts,

17.2 Basic report examples

The following examples of report scripts and their output illustrate most of the features of RPT. Starting with basic techniques, they progress to more advanced report script development.

However, RPT is too extensive to illustrate every conceivable technique. Therefore, you should read the descriptions of the RPT command groups and each command's syntax, later in chapter 17. You can then develop more complex reports.

The examples in this chapter are based on the same three-table personnel database used in chapter 15, SQL-Query/DML Language.

The tables in the personnel database are shown in Figure 17.3..

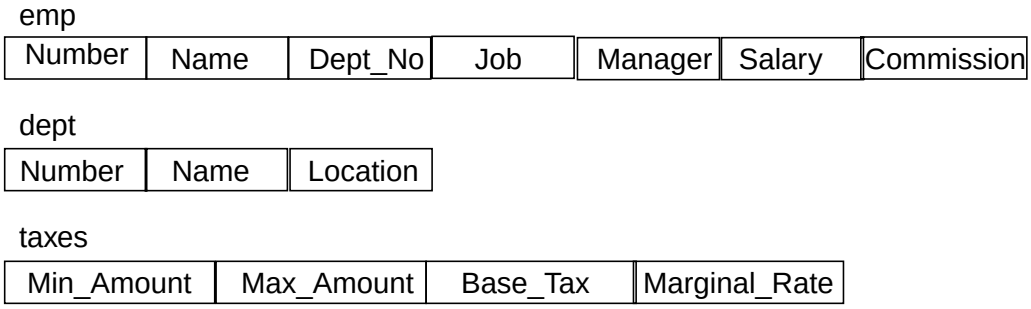


Figure 17.3 Database tables used in RPT examples

The emp table contains the employee's number, name, department number, job, manager's employee number, monthly salary, and yearly commission. The dept table contains the department number, name, and location. The taxes table contains a personal income tax schedule, including the minimum and maximum compensation amounts, the base tax, and the marginal tax rate on income above the minimum amount.

Department listing example

The following subsection introduces a simple report script, and develops it in two steps to produce an example report similar to the previous Example Report. Figures

17.4, 17.5, and 17.6 show three versions of the report, from simplest to most elaborate:

Smith	sapesrep	1500.00	1000.00
Jones	clerk	900.00	0.00
Whittaker	salesrep	2500.00	500.00
Reilly	salesrep	2500.00	1500.00
O'Neil	salesrep	1500.00	150.00
Dugan	salesrep	1650.00	900.00
Schmidt	programmer	2500.00	0.00
Klein	salesrep	1500.00	0.00
Scharf	clerk	$800.00	0.00
Lee	president	7500.00	0.00
Otsaka	engineer	1800.00	0.00
Kawasaki	salesrep	1800.00	1000.00
Dupre	clerk	800.00	0.00
Bleriot	programmer	1100.00	0.00
Moehr	clerk	950.00	0.00
Colucci	salesrep	2500.00	3000.00
Amato	salesrep	2000.00	750.00
Fiorella	clerk	800.00	0.00
Brown	engineer	2000.00	0.00

Figure 17.4 Department Listing #1

Employee Report

Name	Job	Salary	Commission
Smith	salesrep	$18,000.00	$1,000.00
Jones	clerk	$10,800.00	$0.00
Whittaker	salesrep	$30,000.00	$500.00
Reilly	salesrep	$30,000.00	$1,500.00
O'Neil	salesrep	$18,000.00	$150.00
Dugan	salesrep	$19,800.00	$900.00
Schmidt	programmer	$30,000.00	$0.00
Klein	salesrep	$18,000.00	$0.00
Scharf	clerk	$9,600.00	$0.00
Lee	president	$90,000.00	$0.00
Otsaka	engineer	$21,600.00	$0.00
Kawasak	salesrep	$21,600.00	$1,000.00
Dupre	clerk	$9,600.00	$0.00
Bleriot	programmer	$13,200.00	$0.00
Moehr	clerk	$11,400.00	$0.00
Colucci	salesrep	$30,000.00	$3,000.00
Amato	salesrep	$24,000.00	$750.00
Fiorella	clerk	$9,600.00	$0.00
Brown	engineer	$24,000.00	$0.00

Report Complete

Figure 17.5 Department listing #2

D E PA R T M E N T L I S T I N G

For World-Wide Sales, Inc.

10 Administration

Name	Job	Salary	Commission
Scharf	clerk	$9,600.00	$0.00
Lee	president	$90,000.00	$0.00
Jones	clerk	$10,800.00	$0.00
Bleriot	programmer	$13,200.00	$0.00
	Total	$123,600.00	$0.00

20 Eastern Sales

Name	Job	Salary	Commission
Whittaker	salesrep	$30,000.00	$500.00
O'Neil	salesrep	$18,000.00	$150.00
Klein	salesrep	$18,000.00	$ 0.00
	Total	$66,000.00	$650.00

70 Finance

Name	Job	Salary	Commission
Moehr	clerk	$ 11,400.00	$ 0.00
Fiorella	clerk	$ 9,600.00	$ 0.00
	Total	$ 21,000.00	$ 0.00
	Grand Total	$439,200.00	$8,800.00

Average Salary	Minimum Salary	Maximum Salary
$23,115.79	$9,600.00	$90,000.00

Total Employees

19

Report Complete

Figure 17.6 Department Listing #3

Figure 17.4 shows a simple report script that produces a fist of the fields in the input file without formatting, headings, or totals. Suppose you have an input file containing the name, job, salary, and commission for each employee in the database. The report script for Department Listing #1 (Figure 17.4) is as follows:

```
input
   emp_name [string 10],
   emp_job [string 10],
   emp_salary [amount 4],
   emp_commission [amount 4]
detail
   print emp_name, emp_job, emp_salary, emp_commission
end
```

The input chapter tells RPT what the input file looks like. The detail chapter tells RPT how to print each, line. And the end signals the completion of the report script.

The input chapter assigns a name to each field in the input file. You use this name to refer to the field throughout the report script. The field names in the list must be in the same order as the fields in the input file.

There are two ways to specify fields in the input chapter:

1. By giving a name to the field, followed by a *type specification.*
2. By using the database field's *access name.*

To specify fields using a name and a type specification, enclose the type in square brackets. A type consists of a type spec, followed by an optional length as shown:

name [type length]

where the report field name can contain alphabetic characters and the underscore (_). You cannot use Unify DataServer/ELS keywords or reserved words as report field names.

Report field types and lengths are the same as the simple database field types and lengths. Valid types are numeric, float, amount, date, Idate, time, string, and text.

NOTE: binary and comb report fields are not allowed. However, you can use the component fields of a comb field individually.

A length is not required for a date, Idate, or time field, but is required for other types. text fields are restricted to their display length, up to 256 characters. For more information about the length for each field type, see Database Fields, in section 8.1.

To specify fields by access name, use the database field's *l*ong name or implicit name and omit the type specification as shown:

table.field-name

RPT then determines the type and length from the data dictionary.

The short table name must precede the field access name. You can abbreviate the report field name to just its database access name in other RPT chapters, if no other fields in the RPT script have the same access name.

The database field's access name is usually the long field name from the database design.

However, for a field that references a COMB field in another table, each implied field has a separate access name. The implied field's access name is a compound of the referencing field's short database name and the short name of the component field referenced in the other table.

The following is the report script for Department Listing #1 rewritten with database field names:

```
input
   emp.Name, emp.Job, emp.Salary, emp.Commission
detail
   print Name, Job, Salary, Commission
end
```

You can send RPT presorted input from SQL or a custom program by using the sorted option.

When you specify that the input is presorted, you save processing time because RPT doesn't have to resort all the records. In this case, level breaks only occur on presorted fields.

If you use the sorted option, RPT assumes that all fields listed in the input chapter are sorted in the order listed.

To specify the number of presorted input fields, you must set the RPT environment variable, RPTNSITM to that number. When using the sorted option, the input field order and the RPTNSITM setting are extremely important.

For the complete syntax of the input chapter, see Non-Command Groups, section 17.7. For more information on RPT environment variables, see Using RPT With Other Unify DataServer/ELS Tools, section 17.10.

Department listing #2 report script

Two new RPT script chapters are required to add column headings and a "Report Complete" message to the report. The two new RPT script chapters are as follows:

- before report
- after report

Two new clauses are also added:

- column positions column headings and detail line columns
- The using clause contains format instructions for detail lines

The report script for Department Listing #2 (Figure 17.5) is as follows:

```
input
   name [string 10],
   job [string 10],
   salary [amount 4],
   commission [amount 4]

before report
   print 'Employee Report' in column 23
   skip
   print 'Name' in column 1,
      'Job' in column 12,
      'Salary' in column 32,
      ’Commission' in column 44
   print 53[-] in column 1

detail
   print name in column 1,
```

```
   job in column 12,
   salary * 12 in column 32 using '$$$,$$&.&&',
   commission in column 44 using '$$,$$&.&&'

after report
   skip
   print 'report Complete' in column 23
 end
```

The commands in the before report chapter are processed before any detail lines are printed. In the before report chapter, the print command enables you to print character strings, position output explicitly in columns, and print strings of repeating characters. In this case, the repeating characters are 53 dashes.

The skip command leaves blank lines. The default is one blank line. To skip a specified number of blank lines, use the following syntax:

skip nnn

where nnn is the number of lines you want RPT to skip.

In the detail chapter, the fields are positioned under the column headings printed in the before report chapter. The print command lets you print an *expression* with a using format specification. In this case, the expression is salary * 12.

The using format specification, using '$$$,$$&.&&', prints the salary as an amount with a floating dollar sign, a decimal point, and two digits of precision.

For more information about the using option, see the description of the print command in section 17.9.

In the after report chapter, commands are processed after all lines in the input file are printed. In this example, RPT skips one line and prints the message "Report Complete."

Department listing #3 report script

The third report script produces the Department Listing shown in Figure 17.6. This report script uses control break processing to print page headings, column headings, totals, and other statistical information. It sorts the input file by department number in ascending order, and by name in descending order.

The following is the report script for Department Listing #3:

```
input
   dept_no [numeric 2), dept_name (string 15],
   name [string 10], job [string 10],
   salary [amount 4), commission [amount 4)

width 80

sort dept_no, name desc

header
   print 'D E P A R T M E N T  L I S T I N G' centered
   print 'for World-Wide Sales, Inc.' centered
   skip

footer
   skip
   print'--' in column 31, pageno using '&&', '--'

before report
   print 'D E P A R T M E N T  L I S T I N G' centered
   print 'for World-Wide Sales, Inc.' centered
   skip

before dept_no
   need 5
   print dept_no using %-d' in column 1, dept_name
   skip
   print 'Name' in column 1,
      'Job' in column 12,
      'Salary' in column 32,
      'Commission' in column 44
   print 54[-] in column 1

detail
   print name in column 1,
      job in column 12,
      salary * 12 in column 30 using '$$$,$$&.&&'
      commission in column 46 using '$$,$$&.&&'

after dept_no
   need 2
   skip
   print 'Total' in column 22,
      total(salary * 12) in column 29 using '$,$$$,$$&.&&',
      total(commission) in column 46 using '$$,$$&.&&'
   skip
```

```
after report
   need 7
   skip
   print 'Grand Total' in column 16,
      total(salary * 12) in column 28 using '$,$$$,$$&.&&',
      total(commission) in column 45 using '$$$,$$&.&&'
   skip
   print 'Average Salary' in column 1,
         'Minimum Salary' in column 19,
         'Maximum Salary' in column 36
   print avg(salary * 12) column 1 using '$$$,$$&.&&',
      min(salary * 12) column 18 using '$$$,$$&.&&',
      max(salary * 12) column 36 using '$$$,$$&.&&'
   skip
   print 'Total Employees' in column 28
   print count(*) column 28
   skip
   print 'Report Complete' centered
end
```

The sort command specifies the fields to sort by and the order the field are sorted. RPT performs sorting before processing other report commands. The default sort order is ascending, or lowest to highest. You can change the order to descending, or highest to lowest, with the desc keyword.

The header section is processed at the top of each page. You can also use the print option centered in the header section. This option centers the entire print string in the width of the report page.

The footer section is processed at the bottom of each page. You can use the built-in variable pageno in the footer section. The page_no variable contains the number of the current page.

The before dept_no section is a new section. The commands in the before dept_no section are processed before each new group of employees with the same department number.

The need number command lets you make sure there are at least number lines left on the current page. If fewer than number lines remain, a new page is started. In the report script for Department Listing #3, the need number command verifies that there

are at least five lines left on the page. This is enough to print the column headings and one detail line.

The department number is printed using the '%-d' format. This format means print a two-byte left-justified integer.

The detail section is the same as in the report script for Department Listing #2. The after dept_no section, however, lets you print subtotals for each department. This section is processed after each group of employees with the same department number.

The total aggregate function calculates the sum of its expression by adding up the appropriate field or expression for each line in the previous group. For this reason, aggregate functions can only be used in after command groups.

The after report section has been expanded since the Department Listing #2 version:

- The total function at the end of a report calculates a sum, or grand total, of the lines in the input file.
- The other aggregate functions are used at the end of this example as well, to show how they're used. These are avg (average), min (minimum), max (maximum), and count (line count).

Form letter example

The following example shows how you can use RPT to develop a form letter. It also shows you how to use variables and, print conditions to print information based on the data in the input file.

The example report is a form letter from the president to the regional sales offices regarding their sales results for the first quarter of the year. The input file must contain only employees in the regional sales offices. The SQL query must create this file correctly.

The president's form letter lists and congratulates the salesreps who met or exceeded quota. Then, if all salesreps in the office met quota, another compliment is printed. If some of the salesreps in the office fell short, the memo suggests the salesreps increase their efforts to meet quota for the next quarter.

The total sales amount is computed from the commission field. The company pays a 5% commission, so the total sales amount is commission/.05. The form letters produced by the report script are as follows:

----------M E M O----------

TO: Central Sales
FROM: Mr. Lee
SUBJECT: Sales quotas for 1st quarter 1998

As you know, our sales goal for the 1st quarter of 1998 for each salesrep was $10,000. The following salesreps in your department met or exceeded the goal:

Name	**Total Sales**
Reilly	30000.00
Kawasaki	20000.00

Congratulations! Keep up the good work.

The department goal was $20,000.00 and your department had $50,000.00 in sales. Your department exceeded the goal for the 1st quarter, and I want to congratulate everyone responsible.

mrl: jk

----------M E M O----------

TO: Eastern Sales
FROM: Mr. Lee
SUBJECT: Sales quotas for 1st quarter 1998

As you know, our sales goal for the 1st quarter of 1998 for each salesrep was $10,000. The following salesreps in your department met or exceeded the goal:

Name	**Total sales**
Whittaker	10000.00

Congratulations! Keep up the good work.

The department goal was $30,000.00 and your department had $13,000.00 in sales. As you can see, your department failed to reach its quota. I expect everyone will work harder next quarter to meet the objectives.

mrl: jk

----------M E M O----------

TO: Western Sales
FROM: Mr. Lee
SUBJECT: Sales quotas for first quarter 1998

As you know, our sales goal for the 1st quarter of 1998 for each salesrep was $10,000. The following salesreps in your department met or exceeded the goal:

Name	**Total Sales**
Dugan	18000.00
Colucci	60000.00
Amato	15000.00

Congratulations! Keep up the good work.

The department goal was $30,000.00 and your department had $93,000.00 in sales. Your department exceeded the goal for the first quarter, and I want to congratulate everyone responsible.

mrl:jk

The following SQL query is used to produce the input file for the form letters. It selects everyone in the sales offices.

```
select dept.Name, emp.Name, Commission
from dept, emp
where dept.Name = '*Sales*' and
   dept.Number = Dept_NO /
```

The report script required to print the form letters is as follows:

```
input
   dept_name [string 15).
   emp_name [string 101,
   emp.Commission

/* paper size specifications  */
left margin 8
width 80
length 66

sort dept_name

before dept-name
   skip 6
   print ’------- M E M 0 --------' in column 23
   skip 4
   print 'To:', dept_name in column 10
   print 'FROM: Mr. Lee'
   print 'SUBJECT: Sales quotas for 1st quarter 1998'
   skip 2
   print 'AS you know, our sales goal’,
      'for the 1st quarter of 1998,
   print 'for each salesrep was $10,000.',
      'The following salesreps'
   print 'in your department met or exceeded the goal:’
   skip
   print 'Name' in column 8, 'Total Sales' in column 25
   print 35[-] in column 8

detail
   if Commission/.05 >= 10000 then
   print emp_name in column 8, Commission / .05 in column 20

after dept_name
   /* initialize sales totals
   set dept_total to total(commission / .05)
   set dept_goal to $10000.00 * count(*)
   skip 2
   print 'Congratulations! Keep up the good work.'
   skip
   print 'The department goal was',

   dept_goal using ’$$$,$$&.&&’,
   'and your department’
 print 'had', dept_total using ’$$$,$$&.&&’,
   ’in sales. ' no newline

if dept_total >= dept_goal then
begin
```

```
      print 'Your department exceeded the'
      print 'goal for the 1st quarter,',
         'and I want to congratulate'
      print 'everyone responsible.,
   end
   else
   begin
      print 'As you can see, your department'
      print 'failed to reach its quota.',
         'I expect everyone will'
      print 'work harder next quarter to meet the objectives.'
   end

   page

   footer
      print 'mrl:jk'

   end
```

NOTE: You can place documentation comments in an RPT script by starting each comment with /* and ending it with */. Some examples of comments are included in the report script.

17.3 Advanced report techniques

The Department, Listing example in the previous chapter shows you how to write a basic, multiple-level report. The report contains report, page, and column headings. It also contains page numbers, subtotals, totals, minimums, maximums, averages, and counts.

Once you can write this kind of report, you are ready for more advanced techniques. These techniques include more complex arithmetic and Boolean expressions, named expressions, string operators, print conditions, variables, and local functions.

This chapter introduces a database design to track purchase orders and includes the specifications for an advanced report. It describes how to connect a screen form to an SQL query and RPT script. And it shows the complete text of the RPT script necessary to produce the report.

The purchase order database contains the following tables and fields:..

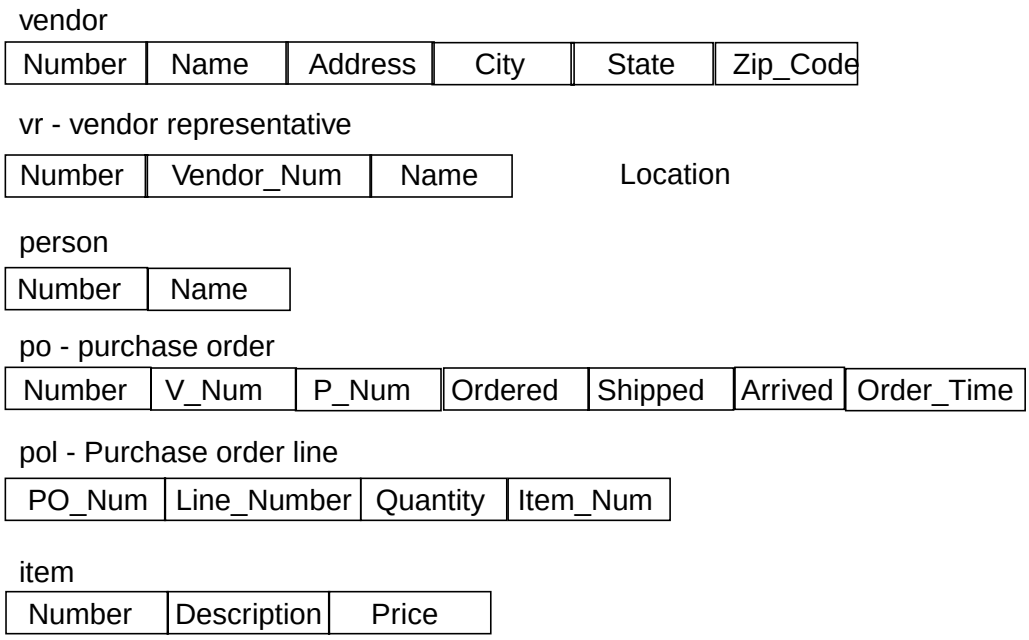


Figure 17.7 Purchase order database tables

The tables in the purchase order database track the vendors, the vendors' representatives, the persons placing the orders, and the purchase orders. Each purchase order consists of a number of lines, and each line refers to an item that is ordered regularly.

The report prints outstanding purchase orders issued within a user-specified date range. The report is divided into two sections. The first section lists the orders not yet shipped by the vendor. The second section lists the orders shipped but not yet received.

The orders in each group are sorted by line number, purchase order number, and vendor number. For each vendor, totals are calculated according to when the purchase order was issued. This can be more than 90 days, 60 to 90 days, 30 to 60 days, or less than 30 days past. If an order was issued more than 90 days ago, it is marked for follow-up action.

Figures 17.8, 17.9, and 17.10 show title page, detail page, and report totals excerpts from the report.

```
                    ************************************************************
                    P U R C H A S E  0 R D E R  S T A T U S  R E P 0 R T
                    ------------------------------------------------------------------------

Purchase orders issued from 10/1/98 to 1/31/99

Friday January 27 1999
Report Run on 01/27/99 at 10:10

*********************************************************************
.
.
.
```

Figure 17.8 Purchase Oorder status report title page

```
Purchase order Status Report 01/27/99 20:10
***orders already shipped***

101   Precision Tool Co.              Contact: Howard Johnson
      2600 West 16th St.
      Alhambra, CA 92155
```

>>>>>Po #1237 Date Ordered: 10/10/99 Date Shipped: 10/17/98

Ordered by: Tony Cartwright on Monday

Item	Qty	Price	Ext
combination pliers	4	$6.89	$27.56
leather mallet	8	$8.75	$70.00
		Total	$97.56

PO #1238 Date Ordered: 12/13/98 Date Shipped: 22/29/98

ordered by: Tony Cartwright on Tuesday

Item	Qty	Price	Ext
3”, Phillips screwdriver	12	$2.35	$28.20
8’ power-lock tape measure	12	$2.69	$32.28
6" slotted screwdriver	12	$2.89	$34.68
		Total	$95.16

Order Summary for Precision Tool Co.

Current 0 Over 30 1 Over 60 0 Over 90 1 Total 2

****

Figure 17.9 Purchase order status report detail page

Grand Totals

Current 6 Over 30 6 Over 60 5 Over 90 2 Total 19

Report Complete

Figure 17.10 Purchase order status report totals page

The title page excerpt displays the date range specified by the user and the day, date, and time of the report.

The detail page excerpt shows the format of the report. A heading shows whether this page is for unshipped or shipped purchase orders. "Greater than" symbols (>>>>>) mark orders issued over 90 days ago, and an aging summary prints for each vendor.

The report totals excerpt shows a final count of purchase orders in each aging category.

To develop this report, you must start with the SQL query that extracts the data. This query joins the six database tables, by selecting purchase orders not yet received but issued between two dates specified by the user. Therefore, you must construct the query so the user can be prompted to enter the dates, and the dates can be substituted when the query runs.

To accomplish this, you place parameters in the query. Then you connect a screen form to the query using Register Screen Form with SQL (chapter 16). The user can then select the screen from a menu and enter the dates. The dates are substituted in the query when it is run by SQL.

A parameter is a place holder in the form $n, where n is a number from I through the number of parameters you want the user to supply. Because you want two dates, your two parameters are $1 and $2. The parameters are placed in the where clause instead of the dates you would normally enter. This part of the where clause is constructed as follows:

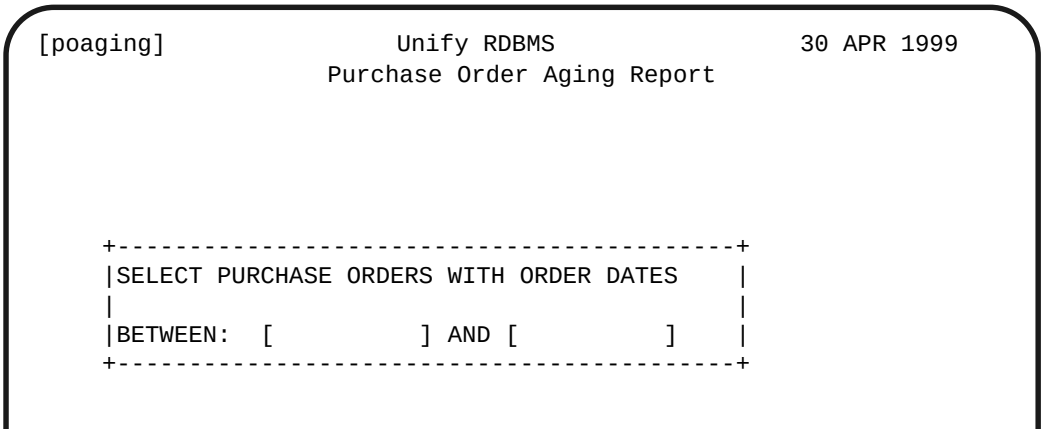
```
where ordered between $1 and $2
```

The entire text of the SQL query reads as follows:

```
lines 0
select vendor.Number, vendor.Name, Address, City, State,
   Zip_Code, vr.Name, po.Number, ordered, Shipped,
   Line_Number, Description, Quantity, Price,
   person.Name
from vendor, vr, person, po, pol, item
where ordered between $1 and $2 and
   Arrived = **/**/** and V Nun = vendor.Number and
   Vendor Num = vendor.Number and
   P_Num = person.Number and PO_Num = po.Number and
   Item_Num = item.Number /
```

The query should be placed in a file and given a name; for example, pq_sql.

The following is a design for the screen form to prompt for the dates.



The value entered for the first screen field is substituted for $1. The value for the second screen field replaces $2. Parameters can be used the same way in report scripts, as shown in the following RPT script:

```
before report
      .
      .
      .
      .
   print 'Purchase Orders Issued from $1 to $2' centered
   print 'Report Run on', today, 'at', hour centered
      .
      .
      .
```

In this report example, you print the purchase order date range and the date and time of the report on the tide page. You can use parameters and the built-in variables today and hour in the before report section to do this. Parameters can be used in the script wherever you want the user to enter a value to be substituted, provided the script still is syntactically correct.

To group the purchase orders in unshipped and shipped orders, you can use sort as in the following expression:

```
sort <dt> Shipped ^= **/**/**
```

The Shipped field is filled in when the vendor says the order has been shipped. So if the date is null, as shown in the expression, the order hasn't been shipped. The

comparison in this expression returns the value 0 (false) if the order hasn't been shipped, and the value 1 (true) if the order has been shipped. Because the value is 0, all unshipped orders are grouped before all shipped orders.

The entire expression is given a name, dt, so you can refer to the control break that occurs with the transition from unshipped orders to shipped orders later in the report.

You can use *local functions* to perform specialized, custom processing. For example, RPT has three built-in local functions that let you print the name of the month and weekday for a date. One of these functions extracts the month number from today's date; for example,1 for January and 2 for February. Then another function translates the number to the appropriate string. The same can be done for the weekday.

The local functions *mdy* and *dow* extract the month and day of the week numbers, and *index* translates the numbers to character strings.

The following commands can be used to set variables to the week and month:

```
set vdate to today
set weekday to <dy> index(dow(vdate), 'Sunday',
            'Monday','Tuesday','Wednesday',
            'Thursday','Friday','Saturday')
set month to <mnth> index(mdy(vdate,0) - 1,'January',
            'February','March','April',,May',
            'June','July','August','September',
            'October','November','December')
print weekday, month
```

If you want to add your own local functions as well, refer to the *Unify DataServer/ELS Programmer's Manual* for more information.

For a report with page titles that change, you can enter the value of a string variable in the header section. Then you reset the variable to different messages, depending on the page being printed.

In the example report, you want the page title to indicate whether the current page contains unshipped or shipped purchase orders. You do this by setting a variable, pg_title**,** in the before report section. This indicates that unshipped orders are being printed. Remember that unshipped orders list first.

When the dt control break occurs, the message changes to indicate that shipped orders are being printed. RPT then skips to the next page. The following are the sections of the report script described so far:

```
sort <dt> shipped  ^= **/**/**, vendor.Number,
                     po.Number, Line_Number
before report
   set pg_title to '*** orders not yet shipped ***'
      .
      .
      .
header
   print pg_title in column 1
      .
      .
      .
before dt
   if shipped  ^= **/**/** then
      begin
      set pg_title to '*** Orders already shipped ***'
      page
      end
```

RPT provides additional string operations. Suppose you want to print string variables, report fields, or constants next to each other without spaces between them. Because RPT normally prints a default space between fields, you can use the direct concatenation operator *(+),* to suppress the default space between fields.

Assuming you have report fields for state and Zip codes, the following print statement positions the state code, leaves two spaces, then positions the zip code in column 11:

```
print State + ' ' + Zip_Code in column 11
```

The direct concatenation operator does not remove trailing blanks from a field. However, this can be inconvenient when you are printing addresses, and you want to eliminate spaces between parts of the address.

For example, in an address, you usually print the city, a comma, a space, the state code, two spaces, and the Zip code, as follows:

```
Sacramento, CA 95825
```

NOTE: The concatenation operators join strings. Therefore Zip_Code must be defined as a STRING in the database design for the purchase order tracking system.

In the database, City is a field of fixed length, 25 characters. Using the direct concatenation operator to print an address line leaves a 25-character wide field for the city. Extra spaces are left when actual name is less than 25 characters. Thus the following print statement:

```
print City + ', ' + State + ’’+Zip_Code
```

gives the result:

```
Sacramento              , CA 95825
```

The *indirect concatenation operator (/+)* eliminates trailing blanks. Thus, the following print statement:

```
print City /+ '.’+ State + ' ' + Zip_Code
```

gives the result:

```
Sacramento, CA 95825
```

Now that you are aware of the features of this advanced report, the following is the complete text of the report script:

```
input
   v_number [numeric 4],
   v_name [string 20],
   Address,
   City,
   State,
   Zip_Code,
   vr_name [string20],
   po_number [numeric 9],
   Ordered,
   Shipped,
   Line_Number,
   Description,
   Quantity,
   Price,
   pe_name [string 20]
```

```
width 80          /* use 8 1/2" by 11" paper */
sort <dt>Shipped ^=**/**/**, v_number,
      po_number, Line_Number

before report
   set pg_title to '*** orders not yet shipped ***'
   set tot30 to 0
   set tot6o to 0
   set tot9o to 0
   set tot_over90 to 0
   skip 10
   print 57[*] centered
   skip
   print 'P U R C H A S E  0 R D E R',
      'S T A T U S  R E P 0 R T' centered
   print 57[-] centered
   skip
   print 'Purchase Orders Issued from $1 to $2' centered
   skip 2
   set vdate to today
   set weekday to <dy> index(dow(vdate),'Sunday',
            'Monday','Tuesday', 'Wednesday',
            'Thursday','Friday','Saturday')
   set month to <mnth> index(mdy(vdate, 0) - 1,'January',
            'February','March','April', 'May',
            'June','July','August','September',
            'October','November','December')
   set day to mdy(vdate,l)
   set year to mdy(vdate,2)
   print weekday, month, day using '%-1d',
      year + 1900 using '%-41d' centered
   print 'Report Run on' , today, 'at', hour centered
   skip
   print 57[*] centered
   page
header
   skip
   print 'Purchase order Status Report' in column 1,
      today in column 31, hour in column 40
   print pg_title in column 1 skip 2
footer
   skip
   print '--' in column 37, pageno using '%ld', '--'

before dt
   if shipped  ^= **/**/**/ then
      begin
```

```
      set pg_title to '*** Orders already shipped ***'
      page
      end

before v_number
   set order30 to 0
   set order60 to 0
   set order90_overg90 to 0

need 10
   print v_number in column 6 using '%-d',
   v_name in column 11,
   'Contact:' in column 42,
   vr_name in column 51
print Address in column 11
print City /+ ',' + State +' ' + Zip_Code in column 11
skip

before po_number
   set old_order to 0
   if Ordered >= today - 30 then
      set order30 to order30 + 1
      else begin
      if Ordered >= today - 60 then
         set order60 to order60 + 1
         else begin
         if Ordered >= today - 90 then
            set order90 to order90 + 1
            else begin
            set orders_over90 to orders_over90 + 1
            set old_order to 1
               end
         end
      end

skip
if old_order = 1 then
   print '>>>>>' in column 1
print 'PO #' in column 6,
   po_number in column 10 using '%-ld',
   'Date Ordered:', Ordered,
   'Date Shipped:', Shipped
Set vdate to Ordered
print 'Ordered by:' in column 6, pe_name /+ 'on', dy
print 'Item' in column 6,
   'Qty' in column 36,
   'Price' in column 44,
```

```
   'Ext' in column 56
print 53[-] in column 6

detail
   print Description in column 6,
   Quantity in column  36 using ’####’,
   Price in column 41 using '$,$$$.&&',


after po_number
   print 'Total' in column 43,
   total(Quantity * Price) using ’$$$,$$$.&&’
   skip

after v_number
   need 7 skip
   print 'Order Summary for ' + v_name in column 20
   skip
   print 'Current ' in column 6, order30 using ’####’,
      'Over 30' in column 20, order60 using ’####’,
      ’Over 60' in column 33, order90 using ’####’,
      'Over 90' in column 46,
      orders_over90 using '####',
      'Total' in column 59,
      order30 + order60 + orderg90 + orders_over90 using ’####’
   ’print 73[*] in column 1
   skip 2
   set tot30 to tot30 + order30
   set tot60 to tot60 + order60
   set tot90 to tot90 + order90
   set tot_over90 to tot_over90 + order_over90

after report
   need 5
   skip
   print 'Grand Totals' in column 32
   skip
   print 'Current ' in column 6, tot30 using '####',
      'Over 30' in column 20, tot60 using ’####’,
      'Over 60' in column 33, tot9O using ’####’,
      'Over 90' in column 46, tot_over90 using ’####’,
      'Total' in column 59, tot30 + tot60 + tot90 +
            tot_over90 using '####'
   skip 2
   print 'Report Complete' in column 32
end
```

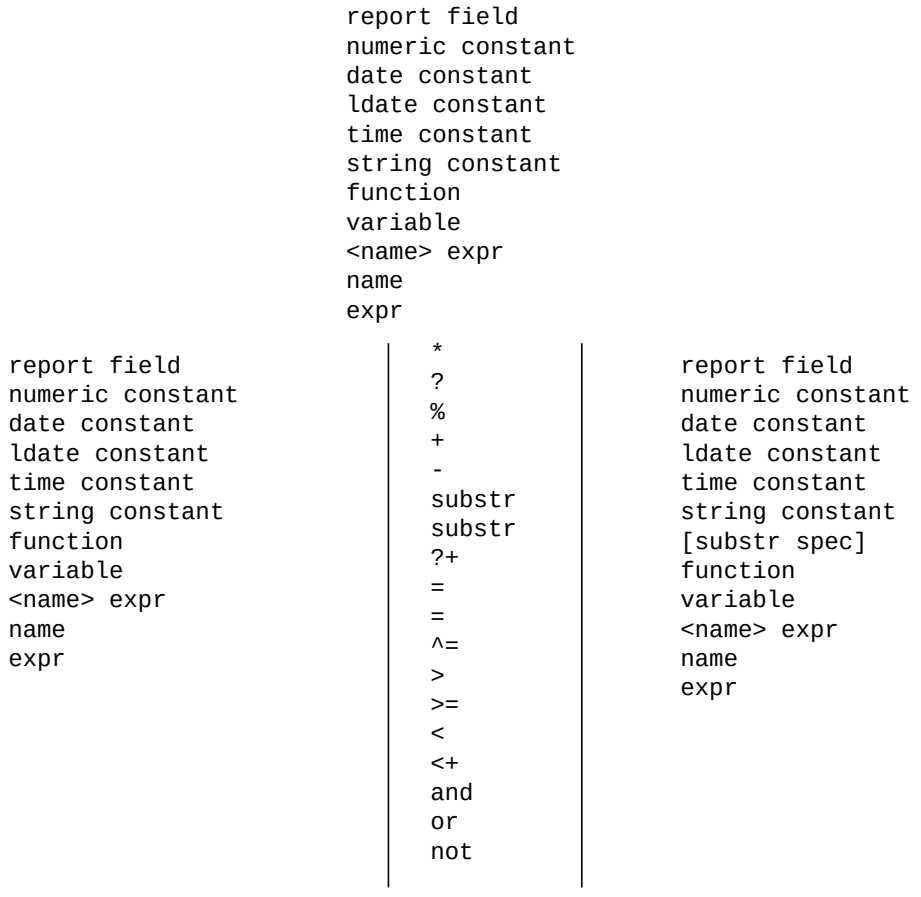
17.4 RPT expression syntax

RPT uses the term expr, to refer to a combination of report fields, constants, variables, functions, and operators. A report field is a name that refers to a particular data field in the input file. You can identify a report field in two ways:

- As table.field
- As name followed by a type specification

In the first method, table is the name of a table in the database and field is the name of a field in that table. In the second method, name is an arbitrary name, and the type specification is a valid data type and length enclosed in square brackets ([ ]). For more information on report fields, see Expression Components, section 17.5.

Expressions tell RPT what to sort, compute, and print. The following is the syntax referred to by expr*:*



An expression or subexpression can be enclosed in parentheses to change the order of evaluation. Without parentheses, the expression operators are evaluated from highest to lowest, as follows:

```
not
*, /,%
+, -, substr, /+
=, ^=, <, >, <=, >=
and, or
```

Operators at the same level are evaluated from left to right.

An expression can be named, then represented by its name in subsequent expressions. Named expressions appearing in a sort command can be used for Control break processing.

Expressions using the comparison operators, =, ^=, <, >, <=, and >=, yield the logical values TRUE or FALSE. These logical values are represented numerically as 1 or 0. The *and, or,* and *not* operators are used to form compound logical expressions.

For information on the use of numbers, dates, times, and the numeric operators (*, /, %, +, and -) in expressions, see Expression Components, section 17.5. That section also describes character string constants, the comparison of strings, and the string operators (+, /+, substr).

RPT recognizes two kinds of functions: `aggregate` and `local.` Aggregate functions perform such operations as total or avg for a group of data values or expressions. Local function results generally depend on the arguments passed to the functions.

You can assign the result of an expression to a variable using the set command. This variable can then be used in subsequent commands where its value is needed.

Named expressions

An expression containing report field names, constants, local functions, and other expression names can be assigned a name using the following syntax:

```
<name> expr
```

You can use an expression name any place in the RPT script where the *`expr`* syntax can be used. Each expression name must begin with a letter. Letters, digits, and underscores can follow to assign a name up to 32 characters.

Suppose item, price, and discount are report fields. The following example illustrates that once an expression is named, the name can be used in subsequent expressions as if it were another field.

```
sort item, <cost> price - discount
detail

print item, <amount> cost + (cost * 0.06) in col 30
print 'CHECK AMOUNT LESS PRICE IS:', amount - price
```

Expressions are named for two reasons:

1. Named expressions appearing in the sort command can be associated with before name and after name command-groups. For more information, see the Control break processing subsection.
2. Expressions can be assigned meaningful names so subsequent commands are concise and easier to read.

An expression consisting of one report field can be named, but it isn't necessary. A report field is a named expression. The field name is the expression's name by default. Therefore, sorted report fields can be associated with before name and after name command-groups.

Logical expressions

A *logical expression* is a combination of comparison operators, logical operators, and values that RPT interprets as TRUE or FALSE. The values can be logical, arithmetic, and string expressions. RPT assigns the value 1 to TRUE logical expressions, and the value 0 to FALSE logical expressions.

The difference between logical and arithmetic expressions is subtle. An expression's type is determined by the operation performed *last* when RPT evaluates the expression. The expression type is logical when the last operation is either logical (and, or, or not) or a comparison (=, ^=, <, >, <=, or >=). The expression type is arithmetic when the last operation is numeric (+, -, *, /, %).

A simple logical expression consists of two values connected by a comparison operator. The following are examples of simple logical expressions:

```
state_code'CA'
I >= 0
a = b
x + 2 <y - 3
```

Compound logical expressions consist of simple logical expressions connected by the logical operators. The following are examples of compound logical expressions:

```
state_code = 'CA' or state_code = 'NY'
a = b and x + 2 < y - 3
```

Logical expressions are used most often in if commands. If the result of the expression is TRUE, RPT processes the group of commands following the then command. If the result is FALSE, RPT processes the group of commands following the else command. For example, consider the following if commands:

```
if 1 = 1 then                    if 1 = 0 then
   print 'TRUE'                     print 'TRUE'
else                             else
   print 'FALSE'                    print 'FALSE'
```

The command on the left prints TRUE, and the command on the right prints FALSE.

Logical expressions can also be used as part of arithmetic expressions. RPT substitutes the numeric value of the logical expression (1 or 0) in the arithmetic expression to perform the evaluation.

NOTE: An arithmetic expression cannot be used in place of a logical expression in an if command.

Comparison operators

Two values can be compared using the comparison operators. Either value can be in the form of a report field, a function, a variable, another expression, or a constant. Figure 17.11 shows the comparison operators:

COMPARISON OPERATORS

=	equal to
^=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

Figure 17.11 Expression comparison operators

Expressions using comparison operators yield the numeric results 0 if FALSE and 1 if TRUE. The table in figure 17.12 shows how comparison operators can be used in expressions. The value of each expression and its TRUE or FALSE interpretations are

shown. fielda and fieldb are report fields containing the numeric values 21 and 0, respectively.

Expression	Result
fielda > fleldb	TRUE
fielda <= fieldb	FALSE
fielda < 34	TRUE
-98.3428 >= flelda	FALSE
fielda + fieldb ^= fielda - fieldb	FALSE
fielda - 21 >= fieldb	TRUE
fielda - (21 >= fieldb)	N/A

Figure 17.12 Comparison expression results

The only arithmetic expression in figure 17.12 is the last expression. The last expression illustrates the difference between logical and arithmetic expressions. It is identical to the logical expression just above it, except for the parentheses that cause the -numeric operator to be evaluated last. Thus RPT interprets it as an arithmetic expression-with a value of 20.

Logical operators

There are three logical operators: and, or, and not. The and and or operators are used to form compound logical expressions. The not operator reverses (or negates) the value of the applicable expression. These three operators can only be used with logical expressions. This means such expressions as 1 and 1 and not 0 are invalid.

RPT evaluates a compound logical expression by determining the value of each subexpression, then combining the results. The tables in figure 17.13 illustrate how RPT evaluates compound logical expressions...

Expression	Result
TRUE and TRUE	TRUE
TRUE and FALSE	FALSE
FALSE and TRUE	FALSE
FALSE and FALSE	FALSE

expression	result
TRUE or TRUE	TRUE
TRUE or FALSE	TRUE
FALSE or TRUE	TRUE
FALSE or FALSE	FALSE

Expression	Result
not TRUE	FALSE
not FALSE	TRUE

Figure 17.13 Logical expression results

The table in figure 17.14 shows how logical operators can be used in compound logical expressions. Again, fielda contains the numeric value 21, and fieldb contains the numeric value 0.

Expression	Result
fielda <= 21 and fielda > 0	TRUE
not (fieldb = 0)	FALSE
flelda > 21 or fieldb < 0	FALSE
fielda > 21 or not fieldb < 0	TRUE
not (fieldb = 0) or (fielda <-- 21 and fielda > 0)	TRUE

Figure 17.14 Comparison expression results

17.5 Expression components

This subsection describes the fields, variables, and constants used in RPT expressions. Valid expressions are formed by using these components with the appropriate operators, described in the previous subsection.

Report fields

A report field, or field, is an RPT expression. The field is a name that refers to a particular data field in the input file. You can specify field as table.field, where table is the name of a table in the database and field is the name of a field within that table. Or you can specify field as an arbitrary name followed by a type specification.

A field can be used wherever the term expr appears in the RPT syntax. The exception is that a report field, or an expression containing a report field, cannot be used in commands appearing in the after report command-group unless it appears in an argument to an aggregate function. This is because all input lines have been processed at the end of the report, and fields have undefined values.

To ensure that a field in an RPT script is named correctly, remember the following rules:

- The name should begin with a letter.
- The name should consist of letters, digits, and underscores.
- The name should not be an RPT keyword. For a list of RPT keywords, see section 17.11.

You can name an expression consisting of one report field, to make the report script more readable. However, you don't have to name the expression because the field name is the name of the expression by default.

Each field has a type that describes the data it contains. The valid report field types are the same as the database field types: numeric, float, date, Idate, time, amount, string, and text. Binary and Combination type report fields are not allowed. However, you can define separate report fields for the component fields that make up a COMB field.

Generally, fields fall into two categories: numbers and strings. Because numeric, float, date, time, and amount fields contain numeric values, they can be manipulated using the numeric operators. string fields can be manipulated using the concatenation operators (+ and /+), and the substring operator (substr).

Numbers, dates, and times

In section 17.4, all the available types of constants are listed in the table defining the syntax of an expression. Constants are explicitly defined numbers, dates, and times, such as -45.30, 12/25/84, and 12:00. Their syntax is consistent throughout Unify DataServer/ELS.

The following describes the comparison of dates and times, as well as the use of the operators /,*,%,+ and - in numeric expressions.

Numeric constants

There are two categories of numeric constants: numbers without a decimal point and numbers with a decimal point. The following is an overview of the syntax of the two categories of numeric constants:

Number without a decimal point:
- Optional preceding minus sign.
- Up to 9 significant digits.

Number with a decimal point:
- Optional preceding minus sign.
- Up to 16 significant digits.

A number that does not contain a decimal point is an integer. It can have up to nine digits. To indicate a negative number, precede it with a minus sign (-).

The following print command shows an expression containing two numeric constants and a field.

```
print -9 * fielda + 123555789 in column 52
```

If a number contains a decimal point, such as a dollar amount or a value with a fractional part, the number can have up to 16 digits. Up to nine digits can appear to

the right of the decimal point. A preceding minus sign can be used to make the number negative.

The following are examples of negative and positive numeric constants containing decimal points:

```
1234567.123456789
987664321.50
-49.99
-53.999898334
-50000.0
0.00
```

RPT can interpret a numeric constant as a numeric, amount, or float type if you append the letter n, a, or f to the constant. This pertains to the explanation of Variable Type Determination later in this chapter.

Date constants

A date constant consists of three values separated by slashes periods (.), or dashes (-). The default format is MM/DD/YY, where MM is the month, DD is the day, and YY is the year. This is the same default format as for an LDATE field data type.

If only two digits are specified, the year value refers to a number of years after 1900. The following are valid dates in the default format:

```
1/1/1       1/1/00      12/31/99       6/3/82
```

The following are not valid default-format dates:

```
0/0/0       13/32/178   3/14
```

You can change the default date format with the DATETP or LDATEFMT environment variables. For a complete description of DATETP and LDATEFMT, see chapter 5.

If DATETP is set, the possible date formats are as shown in Figure 17.15.

DATETP value	DATEFORMAT
AM (American)	MM/DD/YY
EU (European)	DD/MM/YY

Figure 17.15 DATETP settIngs and date formats

DATETP value	DATEFORMAT
IN (International)	YY/MM/DD
US (United States)	MM/DD/YY

Figure 17.15 DATETP settIngs and date formats

If LDATEFMT is set, the possible date formats are as shown in Figure 17.16

LDATEFMT value		
DD/MM/YY	DD-MM-YY	DD.MM.YY
DD/MM/YYYY	DD-MM-YYYY	DD.MM.YYYY
MM/DD/YY	MM-DD-YY	MM.DD.YY
MM/DD/YYYY	MM-DD-YYYY	MM.DD.YYYY
YY/MM/DD	YY-MM-DD	YY.MM.DD
YYYY/MM/DD	YYYY-MM-DD	YYYY.MM.DD
DD/AAA/YY	DD-AAA-YY	DD.AAA.YY
DD/AAA/YYYY	DD-AAA-YYYY	DD.AAA.YYYY

Figure 17.16 LDATEFMT settings and date formats

RPT uses the format specified in LDATEFMT for input and output of all date constants. For input and output of non-constant dates, RPT uses the format specified in either DATETP or LDATEFMT, depending on the data type (DATE or LDATE) of the date field or variable.

NOTE: RPT may not display date constants correctly unless the RPT script is run with the same setting of LDATEFMT (or equivalent month/day/year order) as when the script was written.

RPT has a built-in variable, today, that contains the current month, day and year. today displays in the format determined by the LDATEFMT environment variable. The following is an example of this variable used in a print statement:

```
print today centered
```

today displays as a long date (Idate), with a four-digit year. Date constants also display in the specified long date format.

The following are valid long date constants, when LDATEFMT is set for four-digit years:

```
1/1/1901    6/1/1700    12/32/2099  12/3/1882   01/02/97
```

The.following are valid long date constants for three LDATEFMT settings that specify alphabetic months:

```
3.Jan.2052     21-Dec-1862    6/Mar/95
```

A number of days can be added to or subtracted from a date. However, a date cannot be negative.

Time constants

A time constant must be written in the form HH:MM, where HH is the hour and MM is the minute. The hour value must be from 0 to 23, and the minute value must be from 0 to 59. A time constant is based on the twenty-four hour clock. The following are valid times:

```
0:0      23:59    8:30     12:00
```

The following are not valid times:

```
24:00    23:60    0:99     29:30
```

The time refers to a time-of-day, not the elapsed number of hours and minutes. Times cannot be negative. Nor can they be added. However, you may substract one time from another to obtain the difference between them, measured in hours:minutes.

RPT has a built-in variable, hour, that contains the current hour and minutes, in the format HH.MM. The following is an example of how the hour variable is used in a print statement:

```
print "Printed at:" col 10, hour
```

Comparing dates and times

You can compare two dates or two times using comparison operators. A date cannot be compared with anything other than a date. A time cannot be compared with anything other than a time.

A number of days can be added to or subtracted from a date. The result is a date provided it is in the accepted range, and the result can be compared with another date.

A number of minutes can be added to or subtracted from a time. For example, you can add a number to a time to obtain a new time as follows:

```
1:00 + 60 = 2:00
```

You can also subtract one time from another to obtain the difference in *hour:minutes.*

Numeric operations

Expressions can include the following arithmetic operators:

*	multiplication
/	division
%	modulo
+	addition
-	subtraction

When two numeric type integers are divided, the result is also an integer. The fractional part is truncated. However, you can use the modulo operator (%) to obtain the remainder from the division of two integer values. The result of this modulo operation is also an integer.

You can use parentheses to change the order of evaluation for arithmetic expressions. Otherwise, multiplication, division, and the modulo operation are performed before addition and subtraction. Operators on the same level are evaluated from left to right.

The table in figure 17-17 uses constants to show how arithmetic expressions are evaluated:

Expression	Result
4 % 2	0
5 % 2	1
3 + 2 4	11

Figure 17.17 Arlthmetlc expression resuits

Expression	Result
3 + 2 4 - 6 / 2	8
103 % 10 + 103 / 10	13
6 / 3 * 2	4
6/ (3 % 2)	6
(3 + 2) (4 - 6)	-10
3 * (6 (2 + 1))	6
9-6+ 1.5	4.50
9 + -6 + 1.5 + -3.5	1.00
3.2504 + 1.0096 - 2	2.260000
11/1/83 + 30	12/1/83
3/12/84 - 3/2/84	10
(3/12/84 - 3/2/84) * -18.50	-185.00

Figure 17.17 Arlthmetlc expression resuits

Strings

Strings are report fields of type string or text, character string constants, and any strings formed using string fields and constants with the string operators.

NOTE: RPT displays text fields similarly to string fields. The default display length of a text field is the same as the display length specified for the field in the database design. To display the entire text field, you must include a using clause to format the text field display.

The following subsections describe string constants, string comparisons, and the string operators.

String constants

The word characters refers to printable characters. A character string constant can be specified as a string of characters enclosed in single quotes, or as the repeated character. The repeated character is a character enclosed in square brackets ([1) that is repeated a number of times. The syntax for both methods is as follows:

```
'characters'
count [character]
```

The quoted string describes a literal set of characters. You must enter the string on a single line. The following are examples of quoted string constants:

```
' '                      (the null strng)
'xyz'
'                    '   (string of space characters)
'12 ! A.OK'
```

Headings, underlines, and other unchanging information in a report can be printed by using string constants. A string constant containing no characters (the null string, length zero) is represented by placing two single quotes next to each other with no space between them.

The repeated character syntax (count [character]) specifies the count or number of times you want the character to print. Zero (0) is not a valid count. You must enter one printable character between the brackets.

You can enter long and hard to manage strings using this syntax. The following are examples of repeated character constants:

9[.]	is equal to	'.........'
18	is equal to	' '
20[_]	is equal to	'____________________'
7[']	is equal to	'''''''''''
3 [\]	is equal to	'\\\'

Exactly one printable character can appear between the brackets. Any printable character can be used. Therefore, 64[?], 128[*], and 12[[] are all valid repeated character constants.

Comparing strings

When report information is to be printed under certain conditions, the if command can be used (see section 17.9). The expression following the if keyword in the if command cannot have a string result. However, the expression can consist of string comparisons.

Suppose, for example, you want a special message to print for each occurrence of the state code CA, and statecode also refers to a string of length 2. You can use the following commands:

```
if statecode = 'CA' then
begin
   skip 1
   print 'SPECIAL MESSAGE'
end
```

You can compare two strings using any of the comparison operators: =, ^=, <, >, <=, and >=. Strings are equal only if their contents match and their length, or number of characters, is the same.

The following is a guide on how to order most string characters:

' ' < '0'	'0' < '9'
'9' < 'A'	'A' < 'Z'"
'Z' < 'a'	'a' < 'z'

The first set reads, "The space character is less than the zero character."

When comparing strings, the result is one or zero, as with any comparison. A result of one indicates the comparison statement is TRUE, and zero indicates it is FALSE. Because the result of a comparison is numeric, the result of the following expression is two.

```
1 + ('Hi'  ^=  'Bye')
```

Inexact matches

When comparing strings, you often need to check for a range of characters, or for all the strings with common letters. Special characters can be used as string constants to perform such inexact matches. The following explains how special characters can be used in inexact character matches:

?
: The wildcard character. The question mark matches a single character. For example, if you want to print a special message for all the Smiths and don't know whether the spelling is "Smith" or "Smyth," you can use the string constant Sm?th.

*
: The wildcard string. The asterisk matches a string of characters of any length. This includes zero-length strings, or null strings. For example,

the string constant John* matches all strings beginning with "John," such as:

```
'John'
'John ’
'Johnson'
'John 3:16’
```

The wildcard string (*) helps you compare two strings, when the length of one string is not known, or part of its value must be matched. Suppose you have a report field of type string and length 30. The field name is client, and it contains client names. To print the names of all the clients with the last name "Smith", the if command begins as follows:

```
if client = '*Smith *' then ...
```

The *wildcard string* at the front of the string constant matches first names or initials. The space following Smith ensures such names as "Smithy" or "Smithsonian" do not match. The *wildcard string* after the string constant matches remaining right-end spaces.

[...] The character class. The character class matches a single character that is a member of the class.

For example, all upper case letters can be represented by the character class

```
[ABCDEFGHIJKLKNOPQRSTUVWXYZ]
```

You can specify a range of characters by separating two characters with a dash (-). Consequently, a more convenient way to represent all upper case letters is by [A-Z]. The character class that represents all upper and lower case letters is [A-Za-z]. You can construct other classes similarly.

The character class specifier ([...]) is basically a wildcard character with a defined replacement set. It is often used with the question mark or the asterisk to define a range or set of possible results.

The following string constant matches the state codes from A through F, such as AZ, CA, and FL. The [GKHI] string constant matches the state codes beginning with the indicated letters.

```
'[A-F]?'        '[GKHI]?'
```

Consider the effect of replacing the question marks with asterisks.

You can tell RPT to ignore the meaning of the special characters ?, *, and [ ] in a single-quoted string constant, by preceding them with the backslash character (\). This also applies to the single quote character (') and the backslash character itself.

String operators

Character strings can be manipulated with the three Unify DataServer/ELS string operators: the direct concatenation operator (+), the indirect concatenation operator (\+), and the substr operator. Concatenation is the creation of a new string by joining two strings end to end.

NOTE: String operators cannot be used with text fields.

+ (direct concatenation)

The direct concatenation operator (+) joins two strings and retains trailing blanks. For example, the result of the following expression:

```
'ALPHA' + '   BET   '
```

is the new string:

```
'ALPHA   BET      '
```

The new string contains all the characters from the two original strings, including trailing blanks. However, when you concatenate two strings, you often need to eliminate trailing blanks from the first string. You can use the /+ operator to remove trailing blanks.

/+ (indirect concatenation)

The indirect concatenation operator (/+) removes the first string's trailing blanks before joining the two strings. For example, the result of the following expression:

```
'ALPHA/+ 'BET     ’
```

is the new string:

```
'ALPHABET     ’
```

The trailing blanks are removed from the string on the left. Then the two strings are concatenated. Therefore, the result of the following expression:

```
'ALPHA/+ ’  ’ /+ 'BET'
```

is the new string:

```
'ALPHABET'
```

Remember, two single quotes without a space between them is the null string, or a string with length zero.

substr (substring)

A substring is a portion of a string. A substring can also be the same as the original string. For example, the following are substrings of the string 'AEIOU':

```
’EI'  'IOU' 'A'   "     'AEIOU'
```

Every string contains at least two substrings: the string itself and the null string. However, the null string contains only one substring; that is, the null string itself.

The following is the syntax of the substring operation:

```
string substr [start-end]
```

where the operator is substr. start-end specifies a part of string. start is the starting character position in string, indexed from 1, and end is the ending character position. end must be greater than or equal to start.

Figure 17.18 contains example substring operations and their results.

Expression			Result
'AEIOU'	substr	[1-3]	'AEI'
'AEIOU'	substr	[3-5]	'IOU'
'AEIOU'	substr	[3-9]	'IOU'
'AEIOU'	substr	[6-99]	(null string)
'AEIOU'	substr	[2-2]	'E'
,'AEIOU'	substr	[1-6]	'AEIOU'
'AEIOU	substr	[1-61	'AEIOU'
'AEIOU '	substr	[6-107)	1' '

Figure 17.18 Substring expression results

ALPHA, BET, and AEIOU in the previous examples are constants. But you can use the +, /+, and substr operators with all types of string expressions. The following list shows the string expressions you can use with string operators:

- Report field of type string
- Single-quoted string constant
- Repeated-character constant
- String variable
- String local function
- String expression name
- Expressions with combinations of these and the string operators

Functions

Using a function in an expression is similar to using an expression name. The result of the function replaces the function name in the expression. RPT has two kinds of functions: aggregate functions and local functions.

Aggregate functions calculate results using the value of an expression from many lines of the input file, including results from two or more control break levels in a report.

Local functions calculate results using the values of arguments passed to them. They can only use values that are local to the command-group where they appear as arguments, not values accumulated or monitored during several control break levels.

Aggregate functions

The aggregate functions are count, total, min, max, and avg. These functions can only be used in after name or after report command-groups. Aggregate functions can be used by themselves or as part of an expression. Several aggregates can be used in one expression, and the same aggregate can appear several times.

RPT processes an after name or after report command-group when the associated control break occurs (section 17.6). The value of an aggregate function is calculated using the group of data values corresponding to the after command-group containing the function.

For example, a count function in the after report command-group calculates the total number of records, or lines in the input file, that are processed. The same function in an after name command-group calculates the number of records processed since the last time that after name command-group was run.

The syntax of an aggregate function is as follows:

```
function name (expr [where expr])
```

where the keyword where is an operator. The where keyword indicates that the value of the expression preceding where cannot be considered unless the expression following where is TRUE, or non-zero. Both expressions must return numeric results.

With or without the where operator, the expression inside the parentheses is the aggregate function argument. Local functions, expression names, fields, and constants are valid arguments. Variables and aggregate functions are not.

The following aggregate function calculates the average amount of all amounts greater than seventy-seven dollars.

```
avg(amount where amount > 77.00)
```

In the count function, the expression preceding the where operator is replaced by an asterisk (*). This is because the count function counts records. The number of records is the same no matter which field in the record is counted.

The count function syntax is as follows:

```
count(* [where expr])
```

For an aggregate function, the argument expression is evaluated each time RPT evaluates a record. The total function returns the sum of the resulting values. The min and max functions give the smallest and largest of these values. And the avg function gives their average.

Once the after command-group containing an aggregate function has been processed, the function is reset so the next group of values can be processed.

In the following after command-group example, city and price are report fields or expression names:

```
after city
   print count(*),
      total(l)
   print avg(price),
      total(price)/count(*)
```

NOTE: The two values printed by the first print command are equal. The same is true of the values printed by the second print command.

The table shown in figure 17.19 contains aggregate function expressions using city and price from the previous example.

EXPRESSION

count(*)
avg(price)
2 * avg(price) + 1.75
min (price) + max (price)) /2
total(price >=5.00 and price < 25.00)
max(price where price < 500.00)
min (price where city = 'New York*')
count(* where price >= 500.00)

avg(price where price > 9.00)

Figure 17.19 Aggregate expression examples

Remember, the results of a comparison are 0 for FALSE and 1 for TRUE. Note that one expression finds the lowest price in New York. Another averages the prices over $9.00.

Local functions

RPT comes with three built-in local functions, dow, index, and mdy. You can also develop your own local functions to use with RPT.

A local function is a host language function that can be used with RPT to perform custom formatting, calculating, or database updating from within a report script. For more information about creating your own local functions and linking them with RPT, see the *Unify DataServer/ELS HLI Programmer's Manua*l

A local function accepts a variable length list of arguments, and returns a value of type *char *, short, long,* or *double*. For example, suppose you want to calculate trigonometric functions using data stored in the database. You can use a local function to calculate and return the value to the report.

You can use local functions with just one restriction: The input argument(s) required by the function you want to use must be available when the function is called. Consequently, local functions can be used in sort expressions, as well as in any of the command-groups.

The syntax for calling local functions is as follows:

```
function name ( [arg 1, arg 2, ..])
```

The number of required arguments varies, according to what the specific function expects. Some local functions do not require arguments, but the parentheses must still be used.

Usually, an argument can be any expression that would be valid where the local function is used. The exception is that aggregate functions aren't valid in an argument expression. You can work around this by calculating the aggregate function result, assigning the result to a variable, and then passing the variable to the local function.

The following explains the three local functions included with RPT: *dow, index*, and *mdy*.

dow

SYNTAX dow(date)

USE This local function accepts a report field, constant, or variable of the type DATE or LDATE, and returns an integer indicating the day of the week; for example, 0 = Sunday, 6 = Saturday.

NOTE: dow returns 0 (Sunday) for null Idate expressions and date constants, which are treated as Idates.

The dow function is used with the index local function to print ASCII names for days of the week. For example, to print the day of the week for 1/1/99, use the following expression:

```
index(dow(1/1/99),'Sunday', 'Monday', 'Tuesday',
                  'Wednesday', 'Thursday', 'Friday',
                  'Saturday')
```

index

SYNTAX index(number, string], ..., stringn)

USE This local function accepts a number, followed by a series of constant strings or string expressions. It returns the string indexed by the number, starting from 0. For example, the following call to index returns the string 'def':

```
index(l, 'abc', 'def', 'ghi’)
```

mdy

SYNTAX mdy(date, number)

USE This local function accepts a report field, constant, or variable of the type DATE or LDATE, followed by a number. It returns an integer

signifying the month, day, or year of the date according to number. This is explained in the table shown in Figure 17.20.

Number	Meaning
0	Month number, where 1 means January, 2 means February, etc.
1	Day number, where 1 means the first day of the month, 2 means the second day of the month, etc.
2	The calendar year, e.g., 1987, 2260, 1865, etc.

Figure 17.20 mdy results

The mdy function can be used with index to print dates using words, such as Sunday December 11, 1998.

Variables

The result of an expression can be assigned to a variable using the set command. Variables are useful because once a variable is set to a value, the value can be used in any expression in which a variable is valid.

A variable can be used in an unnamed expression in a command-group command, except in an aggregate function argument. Variables can be used to produce page numbers and section numbers, or to perform complex calculations and character string manipulation.

As with expression names, you choose variable names when you write the RPT script, according to the following rules:

- The variable name should begin with a letter.
- The variable name should consist of letters, digits, and underscores, up to 32 characters.
- The name should not be an RPT keyword. For a list of RPT keywords, see section 17.11.

A variable's type doesn't have to be explicitly declared in the report script. A variable is automatically declared by using it with. The set command. Therefore, you must not set a variable to values of conflicting types. The following are the variable types recognized by RPT:

```
numeric (NUMERIC, AMOUNT, FLOAT)
date
ldate
time
string
text
```

numeric type includes the NUMERIC, FLOAT, and AMOUNT database field types. These field types do not conflict because any numeric value up to nine integer digits can be represented as an amount with zeros to the right of the decimal. Any amount value up to 11 digits to the, left of the decimal can be represented as a float.

The following rules control RPT in determining the type of a numeric variable:

numeric — The expression in each set command that sets the variable returns a numeric value.

amount — The expression in each set command that sets the variable returns either a numeric or an amount value. At least one expression returns an amount value.

float — The expression in each set command that sets the variable. returns a numeric, an amount, or a float value. At least one expression returns a float value.

RPT considers all set commands in a script before choosing a type.

For an expression using a constant, RPT determines the type of the expression result from the way the constant is written. For example, the table in Figure 17.21 shows the

number 1 written in different forms. The table also shows the type dictated by each form.

Constant	Type
1	numeric
1n	numeric
1.	amount
1.0	amount
1.00	amount
1a	amount
1.000	float
if	float

Figure 17.21 Constant data types

If you append the letter n, a, or f to a numeric constant, RPT should be able to interpret the constant as the represented type: numeric, amount, or float.

Initializing variables

Each variable must be initialized to a value using a set command at least once. To set a variable to an initial value, use the following two guidelines:

1. If the variable only needs to be initialized once, place the initializing set command in the before report command-group.
2. If the variable must be initialized before groups of detail information are processed, place the initializing set command in an appropriate before *name* command-group.

For more information on the order in which command-groups are processed, see Control Break Processing, section 17.6.

All variables are initialized to null values before the first command-group begins processing. The table in Figure 17.22 shows the default null initializations:

DISPLAY	MEANING
0	zero if numeric

Figure 17.22 Default null initlalizations

DISPLAY	MEANING
	a null date or Idate
00:00	midnight
”	null string

Figure 17.22 Default null initlalizations

Expression validity table

The expression validity table in Figure 17.23 summarizes the kinds of expressions recognized by RPT and where the expressions can be used in the report script.

Expression Validity Table

An RPT report script can use... / In ...	aggregate functions	local functions	expression names	the name syntax	constants	variables	fields
... the sort command	✔	✔	✔	✔	✔		✔
... an expression following the name syntax		✔	✔		✔		✔
... the header command-group		✔	✔	✔	✔	✔	✔
... the footer command-group		✔	✔	✔	✔	✔	✔
... the detail command-group		✔	✔	✔	✔	✔	✔
... the before report command-group		✔	✔	✔	✔	✔	✔
... a before name command-group		✔	✔	✔	✔	✔	✔
... an after name command-group	✔	✔	✔	✔	✔	✔	✔
... an after report command-group	✔				✔	✔	✔
... an aggregate function argument		✔	✔		✔		✔
... a local function argument		✔	✔	✔	✔	✔	✔

Figure 17.23 Expressions used in RPT scripts

17.6 Control break processing

Control break processing supports the logical grouping of sorted data. Group headings can be printed to label or introduce a group of data items. Calculations can be performed, such as totaling by group or determining group extremes. A special report heading can be printed at the beginning of a report. And, a footer with final totals can be printed at the end of the report.

For example, in a Sales report, sales figures can be totaled by state, by city within state, and by customer within city. Group headings can be used to introduce each new city and state, and a final total of sales can be printed at the end of the report. This arrangement of groups within groups occurs when several fields or values are sorted hierarchically.

The before and after command-groups use control break processing. These groups of commands are run during control breaks. Control breaks occur in the transition between logical groups of data. Usually, unsorted data is not logically grouped. Therefore, only the results of named sort expressions can cause control breaks.

Named sort expressions

RPT supports control break processing by monitoring the sorted values that correspond to each named expression in a sort command. If several expressions are sorted hierarchically, the sort expression that changes least is the most major sort expression. The following example sales report illustrates sort expression hierarchy:

NOTE: The dots in the example sales report represent portions of the report not shown.

SALES REPORT

.

.

.

STATE: CA

CITY: Los Angeles

CUSTOMERS	ORDER#	AMOUNT
ABC Company, Inc	98475	$54,321.99
	423485	$238.67
		$54,560.66
Mom's Auto Repair, Inc.	348661	$98,342.59
	7856322	$23,586.89
	123456789	$55,676.50
		$177,605.98
SALES: Los Angles		$232,166.64
CITY: San Jose		
CUSTOMERS	**ORDER#**	**AMOUNT**
Fixit-Yourself Company	385576	$12,887.50
	1849633	$239.00
		13,126.50
SALES: San Jose		13,126.50
SALES: CA		245.293.14
SALES: NY		$178,385.43
TOTAL SALES THIS REPORT:		$1,003,347.73
AVERAGE CUSTOMER ORDER:		$13,030.49

In the example sales report, the sales input is sorted by state, by city within state, and by customer within city. The expression named state is the most major sort expression, because in the report script, state is the first expression in the sort command. The expression named customer is the most minor sort expression, because in the report script, customer is the last expression name in the sort command.

state, city, and customer are report fields in the example. A report field is the simplest and most common type of named expression. The value of a report field is the value of the corresponding data element in the current line of the input file. The field's default name is the actual name of the field.

More complex expressions composed of several fields, operators, and functions can be assigned a name in the sort command. Then the expression can be referred to by this name throughout of the report.

The following report script contains most of the commands used to produce the example report. It shows the commands necessary for Control break processing:

```
sort state, city, customer

   /* state, city, customer, orderno/*
   /* and price are  database fields

before state
   print 'STATE:', state

before city
   print 'CITY:' in column 6, city

before customer
   need 4
   print 'CUSTOMERS' col 10,
   'ORDER #' col 38, 'AMOUNT' col 54
   print 9[-] col 36, 11[-] col 51
   print customer in col 10

detail
   print orderno col 36, price col 52 using '$$$,$$&.&&'

after customer
   need 2
   print 11[-] in column 51
   print total(price) column 51 using '$$$$,$$&.&&'
   skip I
```

```
after city
   need 2
   print 'SALES:' in column 6. city,
         total(price) column 49 using '$,$$$,$$&.&&'
   print 56[-] column 6
   skip 1

after state
   need 2
   print 'SALES:', state,
   total(price) column 49 using '$,$$$,$$&.&&'
   print 61(_)
   skip 1

after report
   need 4
   print 'TOTAL SALES THIS REPORT:',
   total(price) col 49 using '$$,$$$,$$&.&&'
   print 'AVERAGE CUSTOMER ORDER :',
         avg(price) col 52 using '$$$,$$&.&&'
   print 61[-]
   print 61[-]
```

Control breaks

As RPT processes the sorted values corresponding to named sort expressions, each expression is checked for change in value. When a value is different from the preceding value, a control break occurs.

In the example sales report, a minor control break occurs when all sales to a given customer have been printed, and the next customer is ready to be processed.

A more major control break occurs when the city or state changes. When the state changes, the city and customer change also. When a more major sort expression value changes, minor sort expression values also change.

It is not unusual to print only during control breaks. Many reports require detail information to be processed but not printed. Instead, summary information is printed at control breaks.

Control break processing command-groups

Output such as group headings or totals can be printed during control breaks in a report. The before and after command-groups are used for this purpose.

Before command-groups are used to print section or group headings before processing a group of lines in the input Me with identical sort keys. A before command-group has access to the information in the input line about to be processed. This is how the command-group can identify a control break.

Similarly, an after command-group has access to the information in the group just processed. after command-groups are used to print group totals or summaries after a group.

There are two types of before and after command-groups:

1. The before *name* and after *name* command-groups process during control breaks throughout a report.
2. The before report and after report command-groups process at the beginning and ending of a report.

before AND after name

One before name and one after name command-group can be associated with each named expression specified in the sort command.

When a control break, occurs, the before name and after name command-groups for that level sort expression are processed, as well as the command-groups for any sort expressions lower in the sort hierarchy. For instance, in the example sales report, the input is sorted by state, by city within state, and by customer within city.

Therefore, when a control break occurs because of a change in the city value, RPT processes the after customer commands, followed by the after city commands. Then RPT processes the before city commands, followed by those in the before customer command-group. RPT does not process the command-groups associated with the state values because the state sort expression is more major than the control break being processed.

At a control break, before command-groups are processed according to the sort hierarchy, beginning with the command-group associated with the most major sort expression. The after command-groups are processed in reverse order.

before AND after report

If no sorted values change during the entire report process, two control breaks still occur: at the beginning and at the end of the report. These control breaks are processed differently from control breaks in the body of the report.

Because there is no detail information preceding the control break at the beginning of the report, the before report command-group is processed first. Then existing before name command-groups are processed before any detail information is considered for output. You can use the before report control break to print a descriptive heading identifying the report.

The after report command-group is processed at the end of the report. It usually consists of final totals or overall results. When all after name command-groups have been processed, the after report command-group is processed. There is no detail information to follow, so no before command-groups are processed.

17.7 Non-command-groups

The non-command-group commands can be used once in a report to set up basic report parameters. These report parameters include the input file description, sorting specifications, paper length, and report width and margins.

Non-command-group commands are external to command groups. They usually appear near the beginning of the report script, after the input section. They can appear anywhere in the report script except in the input section or within a command-group. If a non-command-group command follows a command-group, that command-group is terminated. The next command must be either another non-command-group command or the first command in a new command-group.

bottom margin

SYNTAX bottom margin number

USE This command sets the size of the bottom margin. *number* refers to the number of print lines to omit at the bottom of each page. The default bottom margin is 2.

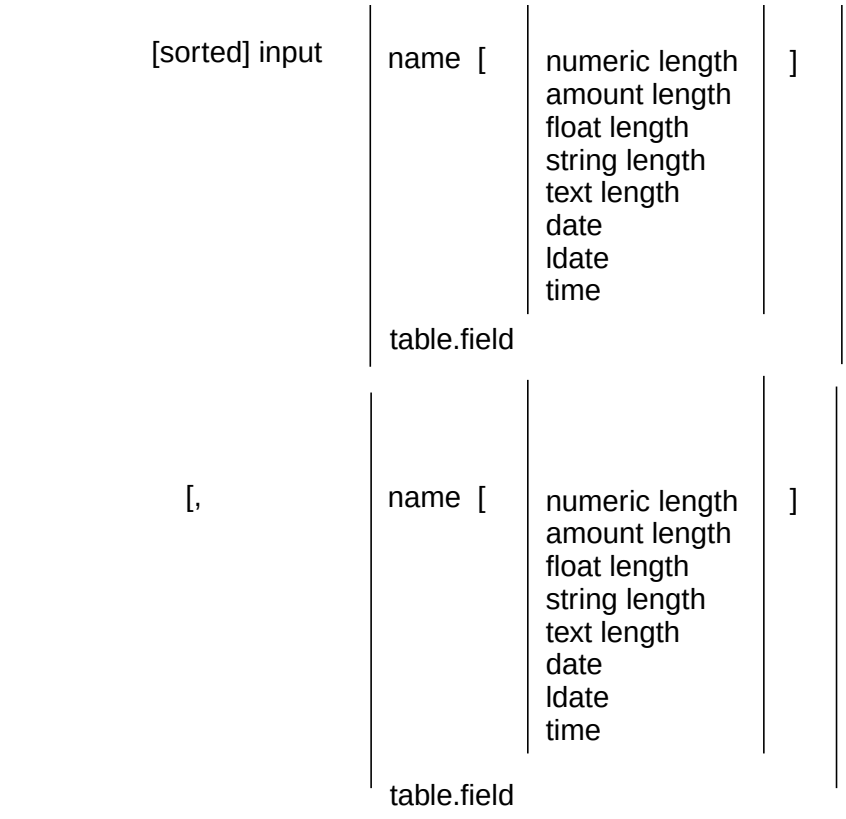
end

SYNTAX end

USE This command indicates the end of the RPT script. RPT ignores all commands after end.

input

SYNTAXs:



USE

This command describes the format of each line in the input file. The report script must have an input section. Each field in the input file must match a corresponding entry in the input section. That is, the first field in the input file must be described by the entry in the input section, the second field by the second entry, and so on.

An input section entry can take one of two forms: a database field access name, or an arbitrary name followed by a type specification.

A database access name is usually the database field's long name. It is the field name returned by the SQL command, fields table.

The type specification after an arbitrary name must be enclosed in square brackets. A type specification is the field type (numeric, float, amount, date, Idate, time, string, or text) followed by the length. The exceptions are date, Idate, and time field types, which have default lengths. Valid values for length are the same as field lengths in the database design.

If you use the database field access name, you need not include a type specification. But you must precede the access name with the name of the table containing the field. RPT looks up the type and length in the data dictionary.

Make sure that a report field is not of type COMB. To use COMB field information, use each component subfield of the COMB field separately.

The sorted option indicates that some input fields are presorted. The environment variable *RPTNSITM* must be set to the exact number of incoming presorted fields. For examples of the input section, see Basic Report Examples, section 17.2.

left margin

SYNTAX left margin number

USE This command sets the width of the left margin. *number* refers to the number of columns to be left blank in the left margin of each page. The default left margin number is 2.

length

SYNTAX length number

USE This command specifies the length of a report page. *number* indicates the number of print lines on one page, including top and bottom margins. The default length number is 66. This assumes you are using line printer paper.

separator

SYNTAX separator 'character'

USE This command describes the field separator character in the input file. Separators are necessary to indicate the end of one field and the start of another. The default separator is the vertical bar (|), used by both RPT and SQL. character can be any single character, including nonprinting control characters.

sort

SYNTAX sort expr [desc]
[, expr [desc] ...]

USE This command determines the report output order. If the sort command is not used, the output is in no particular order. By default, sort arranges the output in ascending order by the specified expressions.

Each expression followed by desc is arranged in descending order by the sort. The sort command can only appear once in a script.

The expressions listed first (from left to right) are major sort expressions. The expressions listed last are minor sort expressions. Sorting on minor expressions doesn't change the order of the major sort expressions. Minor sort expressions determine the ordering of records in groups within major sort groups.

Figure 17.24 illustrates sort sequence for a sort by fieldA, fieldB, fieldC:

```
fieldA valuel           <- major sort expression
   fieldB valuel        <-minor sort expression
      fieldC valuel     <-most minor sort expression
      fieldC value2
   fieldB value2
      fieldC value3
      fieldC value4
fieldA value2
   fie1dB value3
```

Figure 17.24 Sort sequence

A report can only use Control Break Processing efficiently if named expressions are sorted. The expression names can be used to trigger before name and after name

command-groups. However, a named expression can be sorted without being associated with a before name or after name command-group.

For more information about named expressions, see Named expressions in section 17.4 and Expression components, section 17.5.

top margin

SYNTAX top margin number

USE This command sets the size of the top margin. *number* indicates the number of lines to be left blank at the top of each page. The default top margin number is 2.

width

SYNTAX width number

USE This command specifies the width of a report page. *number* indicates the number of columns or print positions available. The left margin is included, so the width need not change if the left margin is changed. For more information, see the left margin command earlier in this section.

The default width number is 132. This assumes you are using line printer paper.

17.8 Command-groups

The command-groups can be used in any order. They are processed in an order determined by RPT, not by the order they appear in the report script.

Each report script must contain at least one command-group. Some reports can be produced using only the detail command-group. Others just require summary information. Summary reports are produced using only the after report command-group.

The following is the syntax of the before report command-group:

```
before report
      command-group commands
```

where before report identifies the command-group. The command-group commands are what RPT processes. If the commands are indented a few spaces, the script's structure is more obvious.

You can begin to write your report script with an outline. Start with the keywords and fill in the commands later. A command-group doesn't have to contain any commands. For example, the command-group identifier header, can be in a script as a reminder to add the appropriate command-group commands later.

A command-group is terminated by another command-group identifier or a non-command-group command, or at the end of the file.

For more information on command-groups, see sections 17.6 and 17.11.

after report

SYNTAX

```
after report
      command-group commands
```

USE This command-group is processed at the end of the report. You can use it to print final totals or report summaries. Therefore, the after

report command-group can only appear in a script once. The processing of this command-group is triggered during the last control break when all after *name* command-groups have been processed.

Generally, only expressions containing aggregate functions, local functions, variables, and constants can be used in after report command-group commands, Expressions can't be named in the after report command-group.

All command-group commands are valid in the after report command-group.

after name

SYNTAX

```
after name
      command-group commands
```

USE

This command-group is processed after a group of detail information is printed. You can use these commands to print group totals or summaries.

name is the name of an expression that appears in the sort command. If an expression is not assigned a name in the sort command, and it is not a database field, the expression cannot be associated with an after name command-group. Each after name command-group must have a different expression name for its name option. For a database field, the field name is the default expression name.

When a control break occurs, the appropriate after name command-groups are processed in reverse order of the named expressions in the sort command; that is, from the most minor to the most major sort expression. For more information on sort expressions, see sort, in section 17.7.

All command-group commands are valid in an after *name* command-group.

before report

SYNTAX

```
before report
      command-group commands
```

USE This command-group is processed at the beginning of the report. It is triggered by the first control break, and precedes all before name command groups. The first line of detail information is evaluated and held to make it available to this command-group while the control break processing takes place.

The before report command-group can only appear in a script once. If this command-group is present, the header command-group is not processed at the top of the first page. Therefore, you can use these commands to print a descriptive heading on the first page of the report. Then you can use the header command-group to print abbreviated headings on the remaining pages of the report.

All command-group commands are valid in the before report command-group.

before name

SYNTAX

```
before name
      command-group commands
```

USE This command-group is processed before a group of detail information is printed. Therefore, you can use before name command-group commands to print group headings or introductory text.

name is the name of an expression that appears in the sort command. If an expression is not assigned a name in the sort command, and it is not a database field, the expression cannot be associated with a before name command-group. Each before name command-group

must have a different expression name for its name option. For a database field, the default expression name is the field name.

When a control break occurs, the appropriate before name command-groups are processed in the order the named expressions appear in the sort command For more information about sort expressions, see sort in section 17.7.

All command-group commands are valid in a before *name* command-group.

detail

SYNTAX

```
detail
      command-group commands
```

USE This command-group is processed for every line in the input file. You can use these commands to print the detail lines of a report and to perform detail calculations.

This command-group can only appear in a script once. All command-group commands are valid in the detail command-group.

footer

SYNTAX

```
footer
      command-group commands
```

USE This command-group's output is printed at the bottom of each page just above the bottom margin.

The number of output lines printed can vary if the if command appears in this command-group. However, for a given report, the footer begins on the same output line on each page. This line is chosen to ensure

that the maximum number of footer output lines print just above the bottom margin.

This command-group can only appear in a script once. All command-group commands except page and need are valid in the footer command-group.

header

SYNTAX

```
header
      command-group commands
```

USE This command-group's output is printed just after the top margin on each page of the report, except the first page. If the before report command-group is not used, this command-group's output is printed after the top margin of the first page as well. This command-group can only appear in a script once.

You can suppress the processing of this command-group during a report by using a page command with the no header option.

All the command-group commands except page and need are valid in the header command-group.

17.9 Command-group commands

The following commands are used only in command-groups. You cannot place a command-group command directly after a non-command-group command or before a command-group identifier.

if

SYNTAX

```
if expr then
   command
[else
   command]
```

where command is either a single command-group command, or a sequence of command-group commands enclosed by the keywords begin and end, as shown in the following:

begin

```
command-group command l
          .
          .
          .
command-group command n
end
```

USE This command varies output or calculations depending on whether the condition described by the logical expression expr is TRUE or FALSE. For more information about logical expressions, see section 17.5.

If the expression is TRUE, or non-zero, the command-group command following the then keyword processes. If the expression is FALSE, or zero, that command is ignored and the command-group command following the else keyword processes. Note that the else keyword and commands are optional.

For a command-group command to be valid in an if command, it must be valid when used alone. Therefore, when if appears in a header or footer, it cannot include the page command, because the page command cannot be used in header or in command-groups.

The need command cannot be used in an if command.

Note that if commands can be nested. For example, the following is a valid if command:

```
if expr1 then
   begin
      if expr2 then
            print field1, field2
      else
            print field3, field4
   end
else
   begin
      if expr3 then
         print field5
   else print field6
   end
```

need

SYNTAX need number

USE This command specifies the number of lines required to continue printing output on the current page.

If there are not enough lines left, output pauses, and appropriate footers print. The printer paper ejects, headers print, and the output continues.

Only one need command can be used in a command-group. The command is evaluated once at the beginning of the command-group, and if there isn't enough space, the page ejects. This command is not valid in the header or footer command-groups, or in an if command.

page

SYNTAX

```
page [, no footer] [, no header]
```

USE

This command causes a page to eject. If specified, the footer prints at the bottom of the current page, and the header prints at the top of the next page.

The no footer option suppresses processing of the footer command-group at the bottom of the current page.

The no header option suppresses processing of the header command-group at the top of the next page, and allows other output to be printed in the heading area. This command is not valid in the header or footer command-groups.

print

SYNTAX

```
print    [expr [[in] col[umn] number] [using 'format']
[,expr [[in] col[umn] number] [using 'format'] ...
[ |no newline| ] ]
  |centered  |
```

USE

This command controls the printing of output. All output is printed using the print command. An expression is evaluated, then its result is printed. Each print command describes one output line or less (see the no newline option). The print command is valid only when used in a command-group.

If the print keyword is used alone, a blank line is printed. Blank lines can also be inserted with the skip command. If the length of an output line exceeds the limit defined by the width command, the line is truncated.

Many reports do not require complicated print commands. For this reason, default spacing and formatting are provided. Expressions can be printed by listing them in a print command. If no column numbers are included, the default spacing ensures that at least one space is left between adjacent expressions.

You can use column numbers for absolute positioning. Use either variable names or numbers to specify column numbers. The print position next to the left margin is column 1.

The database fields used in the following example are price and mrkdwn (type amount), icode (type string, length 3), and iname (type string). The example illustrates default spacing with the print command.

The following command:

```
print price - mrkdwn, icode, iname
```

prints this output:

```
24.95 W26 Widget
```

If you want to print a message starting in column 4, you can use any of these commands:

```
print 'SELL' in Col. 4
print ’ ’, 'SELL'
print 2[ ], ’SELL’
print ’  SELL'
```

NOTE: Three print positions are left blank starting at the left margin.

The column option overrides default spacing. You can use it to make sure two expressions print side by side.

In the following example, the built-in variable pageno prints:

```
print '--' col. 31,
   pageno col 33 using  ’&&’, ’--’ col 35
```

and pageno is automatically incremented by RPT after each page prints. The resulting page numbers would look like --03-- or -48-- in columns 31 through 36.

If you use the no newline option, the next print command's output follows the current command's output on the same line. When print commands contain the no newline option, you can use the if command to vary the output on a single line.

If you use the centered option, the entire print line is centered in the width of the report. You can use this option to print centered titles.

The using option and format templates

Numeric, float, and amount fields usually print in the following default field widths:

```
numeric     9 spaces wide
amount      10 spaces wide
float       16 spaces wide
```

You can change the width of these fields with the using option and a format template.

The format template consists of special characters, one for each position in the field width. Each special character indicates what should print at that position.

The using option special characters

The following describes the special characters for numeric and amount fields. When you read the descriptions, picture the field as formatted and printed from right to left:

#	If there is a digit in this position, print the digit. Otherwise, print a blank. This pads a numeric field with blanks on the left.
&	If there is a digit in this position, print the digit. Otherwise, print a zero. This pads a numeric field with zeros on the left.
*	If there is a digit in this position, print the digit. Otherwise, print an asterisk. For example, this is used

to print amounts on checks that are padded on the left with asterisks.

$ — If there is a digit in this position, print the digit. Otherwise, print a dollar sign. If a dollar sign has already been printed, print a space. This is used to print either a fixed or floating dollar sign.

+ — If there is a digit in this position, print the digit. Otherwise, print a plus sign. If a plus sign has already been printed, print a space.

- — If there is a digit in this position, print the digit. Otherwise, if this is a negative number, print a minus sign. If a minus sign has already been printed, print a space.

(— If there is a digit in this position, print the digit. Otherwise, if this is a negative number, print a left parenthesis. If a left parenthesis has already been printed, print a space.

) — If this is a negative number, print a right parenthesis in this position.

, — If there is a digit to the left of this position, print a comma. Otherwise, print a space.

. — Print a decimal point in this position.

float field format templates

For float type fields, RPT uses a print specification exactly like the C printf function. The print specification has the following format:

```
% [-][minimum field width][.] [precision] f | e |g
```

where the percent sign (%) is required. Options enclosed in square brackets *(*[ ]) are optional. The list of items separated by the vertical bar (|) indicates that you must choose one of the items in the list.

The optional minus sign (-) before the conversion specification indicates that the result is to be left-justified in the field width. The minimum field width is a number indicating the minimum number of print positions in the result. If the result has fewer characters, it is padded to fill the field width.

Precision indicates the number of digits that appear after the decimal point. If precision is 0, no digits print after the decimal point. If there is no decimal point, the number before the conversion character is assumed to be the precision.

The conversion characters are f, e, and g. They have the following meanings:

f	The field is converted to decimal notation in the format *[-]ddd.ddd,* where the number of digits after the decimal point is equal to the precision specification. If the precision is omitted, 6 digits are output. If the precision is explicitly 0, no decimal point is printed.
e	The field is converted in the format *[-]d.ddde+dd,* where there is one digit before the decimal point and the number of digits after is equal to the precision specification. If the precision is omitted, 6 digits are output. If the precision is explicitly 0, no decimal point is printed,
g	The field is converted in format f or e, depending on what provides full precision in minimum space.

Variations of the float format include %-d for a left-justified integer and %ld for a long signed decimal integer.

Format template examples

The table in figure 17.25 illustrates how templates affect output:

Format	Value	Result
'#####'	123	" 123"
'####.##'	0	" "
"#####.&&	0	" .00"
'+++,+++,+++'	23456	" +23,456"
'---,---.&&'	23456.78	"23,456.78"
'---,---.&&'	-2345.67	"-2,345.67"
'($$,$$&.&&)'	2345.00	"$2,345.00"
'($$,$$&.&&))'	-2345.67	"($2,345.67)"
'$##,##&.&&'	1234	"$ 1,234.00
'$**,***.&&'	123	"$***123.00"
'%10.2f'	12.3	" 12.30"
'%10.2f'	123.456	" 123.46"
'%12.4e'	123.456	" 1.2346e+02
'%10.4g'	123.456	" 123.4560"
'%8.4g'	123456789	"1.23e+08

Figure 17.25 Format template examples

The RPT using option also checks for simple errors in the format string. This involves the special character &, which displays leading zeros.

Before printing a value with the format string, RPT checks for illogical combinations involving the & special character. If illogical combinations with & are located, RPT uses a default hierarchy for the format string.

RPT scans the format string from left to right. Once RPT locates the & character, it interprets subsequent special characters as the & character.

For example, when RPT locates the first & character, it replaces characters other than & to the right with the &. The other characters include: #, +, -, *, $. and (. RPT does not change the following characters:

.) ,

The error checking function ensures that the printing of leading zeros is either on (all & characters) or off (no & characters).

Examples of the default format are shown in figure 17.26.

FORMAT	DEFAULT	VALUE	RESULT
'#&&#.##'	'&&.&&'	1234.56	"1234.56"
		1234.5	"1234.50"
		123	"123.00"
		12	"012.00"
		1	"001.00"
		0	"0000.00"
'#&##.&#'	'#&&&.&&'	1234.56	"1234.56"
		1234.5	"1234.50"
		123	"123.00"
		12	"012.00"
		1	"001.00"
		0	"0000.00"
'###.#&'	'###.&&'	123.45	"123.45"
		123.4	"123.40"
		123	"123.00"
		12	"12.00"
		1	"1.00"
		0	".00"
'($#&#.##)'	'($#&&.&&)'	123.45	"$123.45"
		123.4	"$123.40"
		123	"$123.00"
		12	"$12.00"
		1	"$1.00"
		0	"$00.00"
		-23	"($ 23.00)"
'$##&$.&#'	'$##&&.&&'	1234.56	"$1234.56"
		1234	"$1234.00"
		123	"$ 123.00"
		12	"$ 12.00"
		1	"$ 01.00"

Figure 17.26 Format template default examples

using and the CURR Environment Variable

If the CURR environment variable is set, the format specified in *CURR* overrides the using format template as follows:

1. The currency symbol(s) specified in CURR replaces the $ currency symbol appearing in the using format template.
2. The thousands separator specified in CURR replaces the thousands separator appearing in the using format template.

3. The decimal separator specified in CURR replaces the decimal separator appearing in the using format template.

However, the using format template overrides the currency symbol position specified in CURR. The position of the currency symbol, left or right of the amount, as specified in the using format template stays the same. The table in Figure 17.27 illustrates, with example amounts of 100,000 USD's and 100,000 dollars.

IF *CURR* is ...	**AND the using template is ...**	**THEN the output is ...**
'.,2>USD'	'$$$$,$$#.##'	"USD100.000,00"
',.<2$'	'###.##&,&&$'	"100,000.00$"

Figure 17.27 Format template default examples

NOTE: The currency symbol in the using format template must be a single $, regardless of the number of characters specified for the currency symbol in *CURR.* Figure 17.28 illustrates, with an example amount of 340 USD's.

IF *CURR* is...	**AND the using template is...**	**THEN the output is...**
',.2>USD'	'##&.&&$'	"340.OOUSD"

Figure 17.28 Currency Symbol Substitution

using and string fields

You can also format string fields and constants with the using option. Normally, these are printed their full length across the page.

If you want to format a long string field or constant in a column narrower than its length, you can use a format to specify the width of the output column. The string will be broken between necessary words fit it in the specified column width. The format in this case is nr, where n is the width of the output column. The r option specifies a ragged right margin.

For example, there is a string field named, long_string, of length 80. You want to fit the field in a column 20 characters wide, starting at column 40. You must use the following print statement:

```
print long_string using '20r' col 40
```

using and text fields

You can print the contents of a text field with a using clause. RPT treats text fields like string fields.

As for string fields, if you want to format a text field in a report, you can specify a using format for the width of the report column. For example, suppose there is a text field named text field. To print its contents in a 50-character wide column, starting at column 20, use the following print statement:

```
print text_field using '50r' col 20
```

set

SYNTAX set variable to expr

USE This command sets a variable to the result of an expression. *expr is* any expression that is valid in the current command-group. The set command is valid in every command-group. There must be at least one set command for every variable used in a script.

For more information, see *Variables,* in section 17.5.

skip

SYNTAX skip number

USE This command leaves number of lines blank before printing continues. If you omit number, 1 is assumed. Only lines available for printing in the current context are counted. For example, the top and bottom margins are always ignored.

17.10 Using RPT with other Unify DataServer/ELS tools

Unify DataServer/ELS provides built-in interfaces to RPT from SQL and ENTER to simplify application building. RPT can also read any ASCII input file. It then can be used from the shell and by custom user programs.

SQL/Screen form/report interface, section 16.4, describes the SQL-RPT interface as it affects SQL. Similarly, the Register screen form with ENTER, section 15.1, and Using ENTER screens for query by forms, section 15.4, describe how the ENTER-RPT interface affects ENTER. section 17.10 describes what kind of RPT scripts you can write and how to write them using this interface.

This chapter also describes how to run RPT from the shell, and how you can use RPT in programs you write using the host language interface.

SQL by forms and RPT

A convenient feature of SQL by forms (section 16.4) is the ability to format the results of an SQL query using RPT.

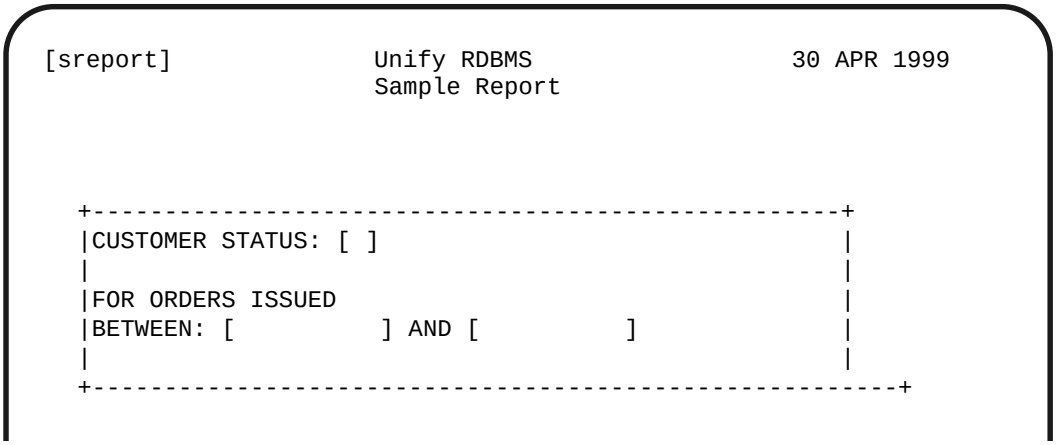
However, for the SQL query and the RPT script to communicate with each other, the fields and expressions you select with SQL must match the input section of the RPT script. The fields in the select clause of the SQL query and the fields in the input section of the RPT script must be in the same order, and of the same data type.

One way to make sure SQL and RPT fields and expressions match is to use the corresponding table.field name in the RPT script for simple fields you select. Use a name followed by a data type and length for fields that are the result of an expression or aggregate function.

Another way to make sure SQL and RPT fields and expressions match is to print the user-supplied selection parameters of the report. For example, suppose you have a report that accepts a date range and a status code from the user. You want a title page on the report to show what has been entered. You can use *parameters* in the RPT script to do this.

Parameters are place holders that mark where the corresponding values from the screen form should be substituted. You can use parameters throughout an RPT script, provided the script is still syntactically correct after the values are substituted.

This example is illustrated by the following screen form:



Parameters are numbered from $1 through the highest screen field number. Note that you are not required to use a particular parameter in a script. You can use all, some, or no parameters.

In this example, the parameter for the status code is $1, the starting date is $2, and the ending date is $3.

To print the title page, you can use a before report section similar to the following:

```
before report
   print 'CUSTOMER STATUS: $1'
   skip
   print 'FOR ORDERS ISSUED'
   print 'BETWEEN $2 AND $3'
```

You can use the same parameters more than once in a script. If you want to print the information at the top of each page, you can include statements similar to the following in a header section:

```
header
   print 'STATUS: $1 FROM $2 TO $3'
```

The previous example shows you how to use parameters in string constants. You can also use parameters in calculations.

Suppose you want to design an aged accounts receivable report that calculates the aging based on a date entered by the user. You can use a series of statements to determine the age of each receivable as in the following:

```
if ar_date >= $1 - 30 then
   /* receivable is current */
else begin
   if ar_date >= $1 - 60 then
      /* receivable is 31 to 60 days old */
   else begin
      if_ar date >= $1 - 90 then
         /* receivable is 61 to 90 days old */
      else begin
         /* receivable is over 90 days old */
      end
   end
end
```

Another way to determine the age of each receivable is to assign the parameter to a variable, as follows:

```
age_date = $1
```

You then can use the variable throughout the script.

ENTER and RPT

ENTER screens enable you to format the results of a query by forms query with an RPT script. If you specify a particular report is to be formatted by RPT, ENTER produces an input file for RPT containing the fields on the screen form. This input file is directed to RPT, which reads the file as its standard input.

To write RPT scripts for this input file, you need to know the order and type of the fields so you can construct the input section. Unify DataServer/ELS provides a command that uses a screen form and writes the corresponding input section for you. The command is called rip. As with other Unify DataServer/ELS commands, rip requires the correct environment variables to be set. For more information about setting environment variables, see chapter 5.

rip

NAME rip-report input section creator

SYNTAX rip screen-name

DESCRIPTION

This command creates the RPT input section for a report script to be run with an ENTER screen. For information about the input section, see Non-Command Groups, section 17.7.

rip prints the correct input section containing the report fields for the given screen in the form table.field. You can use this input section directly in your report script. The following is a sample input section generated by rip:

```
input
   customer.oustomer_number,
   custoner.name,
   customer.address,
   custoner.city,
   customer.state,
   customer.zip_code,
   customer.phone_number
```

USE This command can be used from the A text editor when creating an RPT script. Use the editor command!, followed by rip and the name of the screen form to be associated with the report. vi runs the program and directs the output to the edit buffer.

The correct command line in A is as follows:

```
!rip screen-name
```

User programs and RPT

To use RPT from the shell or with user programs, the appropriate Unify DataServer/ELS environment variables must be set. For more information, see section **S.**

Using RPT from the shell requires two files: an input file and an RPT script file. The input file contains the data you want to format. An input file can come from several

sources, and can either be entered as a file name or piped to RPT. The command syntax for running RPT is as follows:

```
RPT [-r] script [ input_file|   ]
                |              |
```

where script is required. The RPT script file contains the report processor commands to do the report formatting.

The optional -r flag tells RPT to remove the script file when the report is complete. This flag is primarily used by SQL By Forms, which creates and uses a temporary file for the script.

The input file is also optional. This argument is the name of an ASCII file containing the data to be formatted. If you want to pipe the input file to RPT, use the dash (-) instead of the input file name.

If no input file is specified either by name or with the dash, RPT compiles the RPT script file and prints a report. This report lists the number of RPT internal table elements used by the script, and the maximum number of elements in each table.

Each table size can be increased or decreased for scripts that use more elements from one table and less from another. On computer hardware with limited data address space, this enables you to reduce the table sizes and compensate for larger database designs.

To reset a table size, use the corresponding RPT environment variable. These variables and their corresponding tables are explained later in this chapter.

The following are sample command fines used to run RPT from the shell, from a shell script, or from a user program. In all cases, report output is sent to the standard output. Syntax errors and table usage reports are sent to the standard error output.

- The following command line compiles the, script contained in rscript and prints the RPT Table Usage Information report:

  ```
  RPT rscript
  ```

- The following command line formats the data in the file rinput according to the commands in the file rscript:

  ```
  RPT rscript rinput
  ```

- The following command line pipes the results of the SQL query contained in sscript to RPT, and formats the results according to the commands in rscript:

```
SQL sscript | RPT rscript -
```

If you want to use SQL this way, remember to turn off the column headings in the SQL output by using the lines 0 command. You then can use any custom program to pipe to RPT. For more information about the lines command, see SQL Reference, section 16.5.

RPT table usage information report

In the RPT Table Usage Information Report, the terms command and statement are used interchangeably. This report is the result of the following command:

```
RPT rscript
```

And the RPT Table Usage Information report looks something like the following:

```
RPT - Report Processor
Copyright Unify Corporation 1987.

No syntax errors were found in the report script.
RPT TABLE USAGE INFORMATION

Table Name                    Used      Maximum
Expression Nodes              n         400
Variables                     n         250
Constant                      n         250
Commands                      n         256
Print Statements              n         125
Print Items                   n         256
Sort Items                    n         15
Input Items                   n         100
Command Groups                n         25
Set Statements                n         100
If Statements                 n         50
Aggregates                    n         50
Function Calls                n         50
Arguments                     n         100
```

The following describes the internal RPT tables and their corresponding environment variables:

Expression nodes *(RPTNODECNT)*
: This table limits the number of expression nodes allowed in a report script. Each expression in a script uses one *expression node.* The default number of expression nodes is 400. For more information about expressions, see sections 17.4 and 17.5.

Variables *(RPTFLDCNT)*
: This table limits the number of different variables allowed in a report script. The default number of variables is 150. For more information about variables, see section 17.5.

Constants *(RPTCONCNT)*
: This table limits the number of constants allowed in a report script. The RPT constant table includes numeric, float, amount, date, Idate, time, string, and text constants. The default number of constants is 250. For more information about constants, see section 17.5.

Commands *(RPTNCOM)*
: This table limits the number of statements such as sort, after, detail, print, and if allowed in a report script. The default number of commands is 256.

print statements *(RPTNPRINT)* This table limits the number of print statements allowed in a report script. The default number of print statements is 125.

Print items *(RPTNPITM)*
: This table limits the number of print items allowed in a report script. Each comma-separated expression in a print statement uses one print item. The default number of print items is 256.

Sort items *(RPTNSITM)*
: This table limits the number of sort items allowed in a report script. Each comma-separated expression in a sort statement uses one sort item. The default number of sort items is 15.

The *RPTNSITM* environment variable also indicates the number of presorted input fields. This variable should always be set when the sorted option is used with the input statement.

Input items *(RPTNITM)*

This table limits the number of fields allowed in the input section of an RPT script. The default number of fields is 100.

Command groups *(RPTNCGRP)*

This table limits the number of command-groups allowed in a report script. The default number of command-groups is 25. For more information about command-groups, see section 17.8.

set statements *(RPTNSETCL)*

This table limits the number of set statements allowed in a report script. You use set statements to assign values to variables. The default number of set statements is 100.

if statements*(RPTNIF)*

This table limits the number of if statements (commands) allowed in a report script. The default number of if statements is 50. For a description of the if command, see section 17.9.

Aggregates *(RPTNAG)*

This table limits the number of aggregate functions allowed in a report script. Aggregate functions are min, max, avg, count, and total. The default number of aggregate functions is 50. For more information about aggregate functions, see section 17.5.

Function calls *(RPTNCALL)*

This table limits the number of local function calls allowed in a report script. A local function call is when you use a local function in a report script. The function can be one of the RPT, built-in functions or one of your own. The default number of local function calls is 50. For more information about local functions, see section 17.5.

Arguments *(RPTNARGS)*

This table limits the total number of function arguments allowed in a report script. You can pass arguments, or parameters, to local

functions, to repeat an operation on different data. The default number of arguments is 100. For more information about local functions*, see* section 17.5.

sort by key buffer *(RPTPBUFSIZ)*

This table limits the size in bytes of the sort by key buffer. This buffer must be large enough to accommodate the total length of all the sort items' field lengths. The default size of the sort by key buffer is 1024 bytes.

input buffer *(RPTINBUFSZ)*

This table limits the size of the input buffer. This buffer must be large enough to accommodate the total length of all the input items' field lengths. The default size of the input buffer is 1536 bytes.

The table in figure 17.29 summarizes the RPT tables and their corresponding environment variables.

Table	Environment variable	Default value
Expression Nodes	RPTNODECNT	400
Variables	RPTFLDCNT	150
Constants	RPTCONCNT	250
Commands	RPTNCOM	256
print Statements	RPTNPRINt	125
print Items	RPTNPITM	256
sort Items	RPTNSITM	15
Input Items	RPTNITM	100
command-groups	RPTNCGRP	25
set Statements	RPTNSETCL	100
If Statements	RPTNIF	50
Aggregates	RPTNAG	so
Function Calls	RPTNCALL	so
Arguments	RPTNARGS	100
sort by key b	RPTPBUFSIZ	1024
Input key buffer	RPTINBUFSZ	1536

Figure 17.29 RPT tables and environment variables

Setting RPT environment variables

To set an RPT environment variable, use the following syntax:

Using the C shell (csh):

```
setenv variable value
```

Using the Bourne Shell:

```
variable=value
export variable
```

For example, to set the Commands table to 300 and the Input Items table to 40 using the Bourne shell, enter the following:

```
RPTNCOM=300
RPTNITM=40
export RPTNCOM RPTNITM
```

After you have set these environment variables, compile the RPT script Again. The changes you have made to the Commands table and the Input Items table display at the beginning of the RPT Table Usage Information report. This report now displays as follows:

```
RPT:  RPTNCOM set to 300
RPT: RPTNITM set to 40
RPT - Report Processor
Copyright Unify Corpoation 1987

No syntax errors were found in the report script.
RPT TABLE USAGE INFORMATION

Table Name              Used              Maximum
-------------------------------------------------
Expression              n                 400
Nodes
Variable                n                 150
```

```
RPT:  RPTNCOM set to 300
RPT: RPTNITM set to 40
RPT - Report Processor
Copyright Unify Corpoation 1987

No syntax errors were found in the report script.
RPT TABLE USAGE INFORMATION
```

Table Name	Used	Maximum
Constant	n	250
Commands	n	300
Print Statements	n	125
Print Items	n	256
Sort Items	n	15
Input Items	n	40
Command Groups	n	25
Set Statements	n	100
If Statements	n	50
Aggregates	n	50
Function Calls	n	50
Arguments	n	100

17.11 RPT reference summary

With examples, the previous chapters describe how you can use RPT. This chapter summarizes RPT information for quick reference. It lists the reserved keywords used by RPT, summarizes the syntax of commands and command-groups, and explains the error messages produced by RPT.

RPT keywords

The keywords listed in figure 17.30 have specific meaning to RPT and cannot be used for table or field names.

after	amount	and	avg	before	begin
binary	bottom	centered	col	column	comb
count	date	desc	detail	else	end
float	footer	header	hour	if	in
input	idate	left	length	max	margin
min	need	newline	no	not	numeric
or	page	pageno	print	report	set
separator	skip	sort	sorted	string	substr
then	time	to	today	top	total
using	where	width			

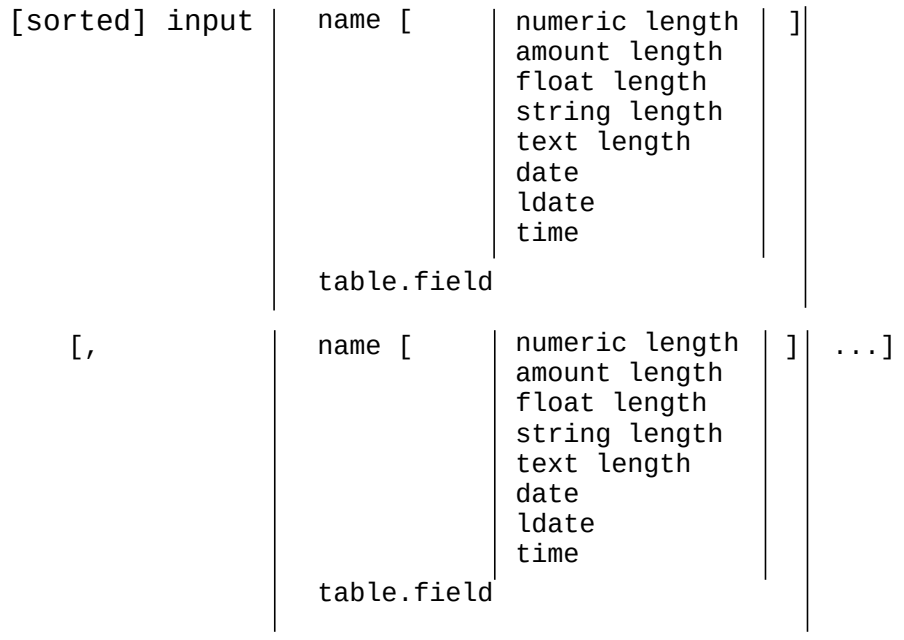
Figure 17.30 RPT reserved keywords

Report scripts that use these words as field names cause syntax errors.

Overview of commands and command-groups

The following is an overview of RPT commands and command-groups. This overview can be used as a quick reference to the commands, command-groups, and Control break processing.

The non-command-group commands, input through sort, and the command-groups are as follows::



length number

width number

top margin number

bottom margin number

left margin number

sort expr [desc]
[, expr [desc] ...]

header command-group commands

footer command-group commands

before report command-group commands

before name command-group commands

detail command-group commands

after name command-group commands

after report *command-group commands*

end

In the RPT script, the input section must be first and the end command must be last. Otherwise, the commands and command-groups can be in any logical order, such as the order they appear in this table.

The length, width, and margin commands are optional and have associated default values. One optional sort command can be used.

You must include at least one command-group in each report script. A command-group is terminated by the next command-group, or by a non-command-group command, such as width. A command-group command can only be used in a command-group. For example, you cannot use the print command directly following a non-command-group command such as sort. RPT reports this as an error.

The before name and after name command-groups can occur more than once. name is a label assigned to an expression in a sort command. A before name command-group often has a matching after name command-group that uses the same name expression. But no two after command-groups can use the same name. Likewise, no two before command-groups can use the same name.

The following commands are used in the command-groups to describe report output.

```
need number

print [expr [[in] col[umn] number] [using 'format']

[,expr [[in] col[umn] number] [using 'format'] ...]

[ | no newline | ]]
  | centered  |

skip number
```

```
page [,no footer] [no header]

set variable to expr

if expr then
   command
[ else
   command]
```

In the if statement, command can be either a single command or a sequence of command-group commands. When command is more than one command-group command, the commands must be enclosed by the keywords begin and end, as follows:

```
begin
      command-group command 1
                .
                .
                .
      command-group command n
end
```

The page and need commands aren't valid in the header or footer command-group.

In command-groups other than the after report command-group, command-group commands can name an expression the first time it is used. The expression can be used again by giving its name. Therefore, expr often implies the following:

```
   expr
[< name >] expr
```

Chapter 18: LST–Listing processor

The Unify DataServer/ELS Listing processor, LST, is a selecting and formatting language that produces sorted lists with totals and subtotals from a Unify DataServer/ELS database. LST is an alternative when the full power of SQL and RPT is not required.

LST has two major components: the selection processor and the listing processor. The selection processor lets you choose records from a file based on selection criteria. The listing processor lets you sort, format, and total the selected records.

The selection processor is a relational query language you can easily use to inquire about a Unify DataServer/ELS database. You can join multiple tables with one query. The syntax of the query does not depend on the specific access methods defined for a Unify DataServer/ELS database. This means that queries can remain the same when the physical or logical database description is changed.

The result of a selection processor query is a selection file, or subset. The selection file contains the relative record numbers of all the records that satisfy the selection criteria.

You can use the listing processor to format the selection file for output. Alternatively, you can use a custom program to format the selection file.

The listing processor is a convenient way to format selection lists, because it has default values for almost every format option. Although it is easy to use, it is also powerful enough to satisfy most output needs.

At the simplest level, you specify the fields to list, and LST assigns default columns, headings, and formats.

At the most complex level, you specify the column, line, and format for each field or expression. You can define sort order, subtotals, and totals. You can also specify multiple-line list and column headings.

If LST cannot satisfy your reporting needs, see chapter 17, RPT-Report Processor, for information about formatting more complex reports.

The following explains the syntax conventions and the types of expressions you can use in LST commands.

LST Syntax conventions

Tbe following conventions are used to describe the syntax of LST commands:

[ ]	Optional item. Square brackets enclosing an item indicate that the entry is optional and can be omitted.
...	Repeat item. The ellipsis indicates that the preceding item can be repeated a number of times.
| |	Choose one item. Vertical bars enclosing a stacked list of items, indicate that one item in the list must be chosen.

Expressions

The term expr is used by the listing processor to refer to a combination of fields, operators, and constants. The syntax of expr is as follows:

expr	+	*expr*
field	-	*field*
number	*	*number*
string	/	*string*
ref-field	=	*start-end*
	!=	*ref-field*
	<=	
	>=	
	<	
	>	
	substr	
	and	
	or	

Expression operators are evaluated in the following order from highest to lowest:

```
*   /   substr
+   -
<   >   <=  >=  =
```

Expressions on the same level are evaluated from left to right. You can enclose expressions in parentheses to change the evaluation order.

The direct concatenation operator, +, can be used to concatenate, or join, two string type elements. Otherwise, only the relational operators and substr can be used with string expressions.

The substr operator extracts a part of a string. Every string contains at least two substrings: the string itself and the null string (length 0). The following is the syntax of the substr operator:

```
string substr [start-end]
```

where the operator is substr. start-end specifies a part of a string where start is the starting position in the string, indexed from 1, and end is the ending character position.

For example, to specify the first 10 characters of a string field named Emp_Number, you enter the following:

```
Emp_Number substr [1-10]
```

For an expression containing a ref-field, or reference field, use the following syntax:

```
field.field ...
```

Reference fields specify fields in tables other than the selected table. The first field must be of the same data type and length as the key of the table containing the second field, and so on. Each field is used to access the next field, up to the final field that is the value of the element.

18.1 Selecting records

To create an LST report, select the appropriate records. You can select records in three ways:

- By using the selection processor.
- By using the query by forms capability of ENTER.
- By using a custom program.

The selection file produced by each of these three methods contains record references to the selected table. The selected table is used by the listing processor to produce the final output.

Running the selection processor

You can run the selection processor from the Menu Handler. You can also run the selection processor from the shell or a shell script.

When run interactively from the Menu Handler or the shell, the selection processor displays the * prompt for you to enter commands. It continues to accept commands until you enter the end command.

When run from a shell script, the selection processor processes the commands in the query-script file until it reaches the end of the file. To run the selection processor non-interactively, use the following syntax:

```
LST query-script
```

Selection processor commands

The following describes the syntax and use of the Selection Processor commands. These commands are as follows:

- select
- remove
- call

- copy
- list
- unlock
- report

select

SYNTAX

select | *subset*
filename | [into string] where expr end

USE The select command selects part of a file or a current subset. The results of the selection are placed in a subset that can be passed to the listing processor, a program, or used in a subsequent select.

If the name following select is not a selected subset, the selection is made from the entire set of records in the specified file. Otherwise, the selection is made from the specified subset. A record is selected when the result of the expression is non-zero for that record.

By default, the results are placed in a subset with the same name as the originating subset or file. If the into syntax is used, the output subset name is specified by string.

remove

SYNTAX remove subset

USE The remove command removes a selected subset. Because select only selects from a file (if there is no subset by the same name), you should remove subsets after using them. Alternatively, you can use call to change the subset name.

call

SYNTAX call subset string

USE The call command changes the name of an existing subset. The specified subset is renamed to string. The renamed subset retains all the characteristics of the original subset.

copy

SYNTAX copy subset string

USE The copy command copies an existing subset. It can be used to perform a reselect while retaining the results of the first selection. The named subset is copied to a new subset with the name string. The copy retains all the characteristics of the original subset.

list

SYNTAX list generic-subset-name

USE The list command lists the existing subsets.

The generic-subset-name has the characteristics of the file specification in the operating system list command.

In generic-subset-name, * matches any number of characters, ? matches any single character, and [...] matches any single character in the specified set. For each subset that matches the specification, the name, table, and number of records are listed.

unlock

SYNTAX unlock fleld password

USE The unlock command allows the selection processor to read a protected fleld. Either the read or write password for the file can be

specified. Attempting to access a locked field without first using this command causes an error.

For more information on protected fields, see section 10.4.

report

SYNTAX report subset

USE The report command invokes the listing processor. The listing processor lets you create simple lists from the selected records.

The subset name is the subset of selected database records that the listing processor must format.

18.2 Listing records

After you select a set of records with the selection processor, ENTER, or a program, you can use the listing processor to format a report. Formatting reports with LST can be simple or complex. At the simplest level, you specify the fields to list, and LST assigns default columns, headings, and formats.

At the most complex level, you specify the column, line, and print format for each field or expression. You can define sort order, subtotals, and totals. You can also specify multiple-line report and column headings.

Running the listing processor

You can run the listing processor in three ways:

1. By using the report command in the selection processor to run the listing processor interactively. Run this way, the listing processor displays the "-" prompt, waiting for you to enter commands. When you are finished, enter the end command to stop the listing processor.
2. By naming the listing processor as a prewritten program to register it with an ENTER screen. To run the listing processor this way, place the listing commands in a file. Then register the file name with the ENTER screen, using Register Screen Form with ENTER (section 15.1).
3. By running the listing processor directly from the shell or from a shell script. To run the listing processor from a shell script, use the following syntax:

```
LST -r selection-file report-script
```

The listing processor processes each command in the *report-script* file in sequence, until it reaches the end of the file.

Listing processor commands

The following describes the syntax and use of the Listing Processor commands. These commands are as follows:

- list

- sort
- total
- go
- print
- nohead
- unlock

list

SYNTAX

```
list [ column number [ line number ] ]
       [ name ] expr  [ using format ]
       [,  [ name ] expr [ using format ] ...  ]
       [ under name ] end
```

USE

The list command prints a series of expressions. You can enter any number of list commands. The expressions in the list command print once for every record in the selection file. The order is determined by previous or subsequent sort commands.

If a name has been defined in a sort command, then you cannot redefine the name. Otherwise, the listing processor uses the name to assign a column heading to the expression.

You can include a slash (/) in the name to indicate that the heading continues on a new line. You can include any number of slashes in a name for a multiple-line, stacked heading.

The default heading for an expression consisting of a single field is the field name. Otherwise, there is no default heading. After naming a heading for an expression, use the heading in subsequent commands instead of the expression.

By default, expressions are assigned columns from left to right on line one. The larger of either the expression output size or the heading size determines the spacing. Default spacing can be altered in two ways:

1. By setting the *line* and *column* absolutely after the list command.
2. By using the under syntax to indicate that the expressions are to appear below a previously-defined expression.

The using Format

Numeric, float, and amount fields are normally printed using a default format. You can change the print format of these fields with the using syntax and a template composed of special characters. Include one special character for each position in the field width.

The special character at each position in the template indicates what should print at that position. The special characters for numeric and amount fields are as follows:

#	If there is a digit in this position, print the digit. Otherwise, print a blank. This pads a numeric field with blanks on the left.
&	If there is a digit in this position, print the digit. Otherwise, print a zero. This pads a numeric field with zeros on the left.
*	If there is a digit in this position, print the digit. Otherwise, print an asterisk. For example, you can use this character when printing checks and you want the amounts padded with asterisks on the left.
$	If there is a digit in this position, print the digit. Otherwise, print a dollar sign. If a dollar sign has been printed, print a space. You can use this character to print either a fixed or floating dollar sign.
+	If there is a digit in this position, print the digit. Otherwise, print a plus sign. If a plus sign has been printed, print a space.
-	If there is a digit in this position, print the digit. Otherwise, if this is a negative number, print a minus sign. If a minus sign has been printed, print a space.
(	If there is a digit in this position, print the digit. Otherwise, if this is a negative number, print a left parenthesis. If a left parentheses has been printed, print a space.

) If this is a negative number, print a right parenthesis in this position.

, If there is a digit to the left of this position, print a comma. Otherwise, print a space.

. Print a decimal point in this position.

For float type fields, LST uses a print specification exactly like the C language printf function. For more information about printf, refer to any C Programmer's Manual.

The print specification has the following format:

% [-] [*minimum field width*] [.] [*precision*] f I e I g

where the percent sign (%) is required. Items enclosed in square brackets ([ ]) are optional. The list of items separated by the vertical bar (|) indicates that you must choose one of the items in the list. These items, f, e, and g, are the conversion characters.

The optional minus sign (-) before the conversion specification indicates that the result is to be left-justified in the field width.

The minimum field width option specifies the minimum number of print positions in the result. If the result has fewer characters, it is padded to fill the field width.

The precision option specifies the number of digits that appear after the decimal point. If precision is 0, no digits print after the decimal point. If there is no decimal point, the number before the conversion character is assumed to be the precision.

The conversion characters are described as follows:

f The field is converted to decimal notation in the format [-]ddd.ddd, where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are output. If the precision is explicitly 0, no decimal point is printed.

e The field is converted in the format [-]d.ddde+dd, where there is one digit before the decimal point. The number of digits after the decimal point is equal to the precision specification. If the precision is missing,

6 digits are output. If the precision is explicitly 0, no decimal point is printed.

g — The field is converted in the format of f or e, whichever gives full precision in minimum space.

The examples in Figure 18.1 illustrate how templates can affect output.

Format	Value	Result
'#####'	123	" 123"
'#####.##'	0	" "
'#####.&&'	0	" .00"
'+++,+++,+++'	23456	" +23,456"
'---,---.&&'	23456.78	" 23,456.78"
'---,---.&&'	-2345.67	" -2,345.67"
'($$,$$&.&&)'	2345.00	" $2,345.00"
'($$,$$&.&&)'	-2345.67	"($2,345.67)"
'($##,##&.&&)'	1234	"$ 1,234.00"
'$**,***.&&'	123	"$***123.00"
'%10.2f'	12.3	" 12.30"
'%10.2f'	123.456	" 123.46"
'%12.4e'	123.456	" 1.2346e+02"
'%10.4g'	123.456	" 123.4560"
'%8.49g	123456789	"1.23e+08"

Figure 18.1 Format Template Examples

sort

SYNTAX

```
sort [reverse]
          [uniquely]
          [name] expr [, [name]  expr ...] end
```

USE

The sort command determines the output order. If the sort command is not used, the output not ordered. By default, sort arranges the output in ascending order according to the specified expressions.

If the reverse syntax is used, the output order is reversed. The expressions are listed according to decreasing sort importance.

The optional name argument assigns the expression a name that can be used in total and list commands. If a name has been assigned and the expression is listed or totaled, the name is used as a column heading.

The uniquely syntax specifies that duplicate lines should not be output. An entire line must be identical to be suppressed, not just the sort expression. As a result, each line in the list contains a unique set of sort expressions.

total

SYNTAX

```
total │ [name] expr │
      │   name      │ [using format],...
      │    expr     │  [by name [, name] ] end
```

USE

The total command outputs expression totals and subtotals. By default, the named expressions are totaled for the entire set of selected records. The by syntax produces totals at level breaks. The names specified after by are the names of expressions that have previously appeared in sort statements. A total is produced for each change in value for these expressions.

LST supports up to five sort expressions in a sort command. Correspondingly, LST supports a maximum of five break level subtotals, as well as grand totals.

If the expression was named previously, only that name should be used. If the expression was named in a previous list statement, the output column and format are the same as that specified in the list

statement. If the expression did not appear in a list statement, the name can be used to give the expression a column heading.

A slash (/) can be included in the name to indicate that the heading continues on a new line. Any number of slashes can be included in a name.

If no name is specified and the expression consists of a single field, the default heading is the field name. Otherwise, there is no default heading.

If the expression was not named in a previous list statement, the using syntax can be used to define the output format.

go

SYNTAX go

USE The go command displays the list on the standard output, usually the screen. The output stops every 23 lines for user input. The column headings are reprinted after each page unless the nohead command was previously issued.

You should probably use print in LST scripts named as the formatting parameters for ENTER screen reports (section IS). Otherwise, the specified report will print without a heading.

print

SYNTAX print report-heading

USE The print command sends the list to the standard output device, usually the printer. The output includes page numbers, date, time, and report heading, as well as the specified column headings. The report-heading is printed on every page.

A slash (/) can be included in the report-heading to continue the heading,Qn a new line. Any number of slashes can be included in the report-heading to indicate a stacked, multiple-line heading.

You should probably use print in LST scripts named as the formatting parameters for ENTER screen reports, so the reports will print with headings. For more information about parameters and ENTER screen reports, see chapter 15 in this manual and chapter 11 in the *Unify DataServer/ELS Developer's Tutorial Manual.*

nohead

SYNTAX nohead

USE The nohead command run before a go command suppresses column headings on the output. This can be used to dump data from the database to ASCII files without extraneous header information.

unlock

SYNTAX unlock field password

USE The unlock command reads a field protected by field level security. Either the read or write password can be specified to unlock a field. If an attempt is made to access a protected field in a report without using this command, an error message displays.

Appendix A: Field and internal data types

Figure A.1 illustrates Unify DataServer/ELS field data types and their possible internal types.

Field		Internal	
Data Type	**Display Length**	**Data Type**	**Storage length**
NUMERIC	1-4		
DATE	n/a	short	(*)
TIME	n/a		
NUMERIC	5-9		
AMOUNT	1-7	long	(*)
LDATE	n/a		
FLOAT	199	double	(*)
AMOUNT	8-11		
STRING	I - 256	char	1-256
TEXT	1-256	char	unlimited
BINARY	n/a	char	unlimited
COMB	n/a	n/a	

Figure A.1 Table of valid field data types

(*) Internal storage lengths are machine-dependent. Many machines use 16-bit shorts, 32-bit longs and 64-bit doubles.

Field data types and lengths describe how field data displays. For information on Data Base Field Types, see Introduction to Data Base Design, in chapter 7. Internal data types represent how much storage space the field data type uses internally.

A.1 Changing field data types and lengths

When changing a field data type or display length, you must consider the associated internal data type. If you change the internal data type, you are changing how much storage space is used for the field data.

If you make the internal data type smaller, any data that exists in that field is forced to fit in a smaller storage space.

For example, changing a NUMERIC 5 to a NUMERIC 4, converts data from an internal data type long to a short. Similarly, changing a STRING 100 to a STRING 10, takes data from in a 100-character storage space and stores it in a 10-character space. Obviously, either move can cause data loss.

Therefore, it is safer to increase the size of a field, or change its field data type so that the internal data type either increases or stays the same.

Another important consideration before changing a data field type is data type compatibility. Two'field data types are compatible if you can change the data type of the first field to the data type of the second field without losing data.

Figure A.2 shows the categories of compatible field data types.

Category	Compatible data types
Number field data types	NUMERIC, FLOAT, AMOUNT
Date field data types	DATE, LDATE
Text field data types	STRING, TEXT

Figure A.2 Compatible field data types and categories

You can change field data types in a category, but never outside it. The examples of compatible data type conversions are AMOUNT to NUMERIC, NUMERIC to FLOAT, DATE to LDATE, or STRING to TEXT. However, AMOUNT to DATE, or STRING to FLOAT are not compatible data type conversions.

If you change the data type from one format to an incompatible format, the data is misinterpreted, converted, and stored by the system incorrectly. Consequently, the data for this field is damaged and unreadable.

Therefore, to avoid loss of data when changing a field data type or display length, consider the following rules:

1. Only increase the field display length. For example:

```
NUMERIC I    ->  NUMERIC 4
AMOUNT 5     ->  AMOUNT 7
STRING 20    ->  STRING 50
```

2. Never decrease the internal length. For example:

```
NUMERIC 7 (long)     ->     NUMERIC 3 (short)
AMOUNT 10 (double)   ->   AMOUNT 7 (long)
```

3. Only change a field data type to a compatible data type. For example:

```
DATE     ->          LDATE
NUMERIC     ->     AMOUNT
NUMERIC     ->     FLOAT
AMOUNT     ->      FLOAT
STRING     ->      TEXT
```

The following outline is a guide to changing field data types and lengths. Safe changes mean no data loss should occur. Unsafe changes can cause damage to data.

Refer to the table in figure A. 1 when using the following guide.

Safe Changes

Changes in format-compatible data types. Safe changes make sure the internal data type is increased in length or stays the same:

```
AMOUNT (long)    ->          AMOUNT (long)
AMOUNT (long)    ->          AMOUNT (double)
```

AMOUNT (long)	->	NUMERIC (long)
AMOUNT (long)	->	FLOAT (double)
AMOUNT (double)	->	FLOAT (double)
FLOAT (double)	->	AMOUNT (double)
FLOAT (double	->)	FLOAT (double)
NUMERIC (short)	->	NUMERIC (short)
NUMERIC (short	->	NUMERIC (long)
NUMERIC (short)	->	AMOUNT (double)
NUMERIC (short	->)	FLOAT (double)
NUMERIC (long)	->	AMOUNT (long)
NUMERIC (long)	->	AMOUNT (double)
NUMERIC (long)	->	FLOAT (double)
DATE (short)	->	LDATE (long)
STRING (char[n])	->	STRING (char[n or >n])
STRING (char[n]	->	TEXT (unlimited)
TEXT (unlimited)	->	TEXT (unlimited)

where:

n = total number of characters

>n = greater than n

Unsafe Changes

Changes that move data from larger internal storage spaces to smaller ones:

AMOUNT (double)	->	NUMERIC (short)
AMOUNT (double)	->	NUMERIC (long)
AMOUNT (double)	->	AMOUNT (long)
FLOAT (double	->)	NUMERIC (short)
FLOAT (double)	->	NUMERIC (long)
FLOAT (double)	->	AMOUNT (long)
NUMERIC (long	->	NUMERIC (short)
LDATE (long)	->	DATE (short)

STRING (char[n])	->	STRING (charf<n])
TEXT (unlimited)	->	STRING (charf<n])

where:

n = total number of characters

<n = less than n

Changes in field data type to a non-compatible format:

number	<->	DATE, TIME, LDATE, STRING, TEXT,BINARY
DATE, LDATE	->	number, TIME, STRING, TEXT, BINARY
TIME	->	number, DATE, LDATE, STRING, TEXT, BINARY
STRING	->	number, DATE, LDATE, TIME, BINARY
TEXT	->	number, DATE, LDATE, TIME, BINARY
BINARY	->	number, DATE, LDATE, TIME, STRING, TEXT

where:

number = number data types (AMOUNT, NUMERIC, FLOAT)

Appendix B: Unify DataServer/ELS reserved words

The following keywords have specific meaning for Unify DataServer/ELS . Therefore, you cannot use keywords for table or field names.

after	amount	and	asc	avg	before
begin	between	binary	bottom	by	centered
col	column	comb	count	date	delete
desc	detail	edit	else	end	fields
float	footer	from	group	having	header
help	hour	if	in	input	insert
into	is	Idate	left	length	lines
margin	max	min	need	newline	no
not	numeric	or	order	page	pageno
print	report	restart	select	separator	set
skip	sort	sorted	start	string	substr
sum	tables	text	then	time	to
today	top	total	unique	unlock	using
where	width	write			

Figure B.1 Unify DataServer/ELS reserved words

Likewise, the names of Direct HLI functions cannot be used for table or field names because the Unify DataServer/ELS preprocessor converts each function name into its identifying number as given in file.h. For a list of functions, see the *Unify DataServer/ELS Direct HLI Programmer's Manual.*

Appendix C: Custom message control

Appendix C contains information for customizing your database application. You can customize your application as follows:

- Using Custom message control, you can change the messages that Unify DataServer/ELS displays while running.
- You can set the CURR environment variables to specify currency display formats.
- You can set the DATETP and LDATEFMT environment variables to specify date display formats.
- Using Reconfigure data dictionary, you can modify the Unify DataServer/ELS master data dictionary to change Unify DataServer/ELS system menu headings and other system characteristics.

C.1 Custom message control

Custom message control enables you to modify the messages displayed by Unify DataServer/ELS . You can change a message to display custom instructions for the user. For example, a message can tell the user the following:

Contact the System Administrator at Extension 326.

You can even change all the messages to another language.

All Unify DataServer/ELS messages are stored as ASCII text in the following two files:

- unifymsgs
- localmsgs

Both unifymsgs and localmsgs are located in the Unify DataServer/ELS library (the lib directory) after you install Unify DataServer/ELS . The Unify DataServer/ELS library is the directory specified in the Unify DataServer/ELS environment variable. For more information about setting the Unify DataServer/ELS environment variable, see chapter 5.

You can edit the ASCII text messages in unifymsgs and localmsgs, using a text editor such as vi. Find the lines that contain the messages you want to change, and replace the existing messages with the new ones.

A message can be up to 140 characters long. Enter the message on a single line, without wrapping. The message can include special characters, such as \t for a tab or \n for a new line. You can also include control characters in the message. For example, you can use a CTRL G to ring the bell when the message displays.

!WARNING Edit only the text messages in unifymsgs and localmsgs. Do not change the numbers preceding the messages. Unify DataServer/ELS uses these numbers to access the messages.

Formatting currency and date displays

In addition to changing messages, you can set the CURR, DATETP, and LDATEFMT environment variables to display currencies, dates, and times in the correct format. For more information about the CURR, DATETP, and LDATEFMT environment variables, see chapter 5.

The following is a brief description of these three environment variables:

CURR — The display format for amounts. Up to seven characters can be entered to define the currency amount format. The first three characters are required and the last four characters are optional.

The default currency format is comma-separated thousands, with a decimal point and two decimal digits, as follows:

```
CURR=",.2"
```

DATETP — The format in which to accept and display dates from January 1, 1900 to December 31, 1999. DATETP can also be used to change all brackets that display on menus and screen forms to parentheses.

The default date format is MM/DD/YY, where M stands for month, D stands for day, and Y stands for year. The default format also displays brackets on menus and screen forms.

LDATEFMT — The format in which long dates (four-digit years) from October 1, 1752 through December 31, 9999 are accepted and displayed. If LDATEFMT is not set, the default format is MM/DD/YY.

C.2 Reconfigure data dictionary

The Reconfigure data dictionary utility enables you to modify several data dictionary attribute tables. You can increase the number of tables, fields, screens, programs, B-trees, field relations, group and employee entries, and volumes. Reconfigure Data Dictionary is an essential tool for the developer who needs greater control-and design flexibility for a Unify DataServer/ELS database application.

Just as you can increase the Expected number of records for any of the tables you create in your application database, you can increase the Expected number of records for total tables allowed or total fields allowed.

Section C.2 explains what a data dictionary is, as well as the difference between the master data dictionary and an application data dictionary. This chapter also explains how to customize the master data dictionary, calculate the Expected Number of Records in master data dictionary tables, and reconfigure the master data dictionary after modifying it.

For information about changing the Expected number of records for an application database, refer to section 8.2.

What is a data dictionary?

A data dictionary contains a schema describing the design of a database. The schema includes the total number of records and fields allowed in the database, how these records and fields relate to each other, and the location of these records and fields.

The structure of Unify DataServer/ELS is based on a schema contained in the Unify DataServer/ELS master data dictionary. When you create a data dictionary for an application, Unify DataServer/ELS uses the master data dictionary to create a template data dictionary file that Will contain the schema for the application.

The master data dictionary

The Unify DataServer/ELS master data dictionary file, master.dd, is located in the Unify DataServer/ELS library (the lib directory). The master data dictionary contains the following information about the structure and design of the Unify DataServer/ELS database:

- The schema elements (tables, fields, field relations)
- Screens
- Programs (menus & executables)
- B-trees
- Data security (groups and employees)
- Storage information (in multi-volume databases)

The actual data for master.dd is kept in sunify.db. sunify.db contains the total number of tables and fields allowed, the names of system menus and screens and the programs that control them, the total number of B-trees and volume definitions allowed, and other information.

You can compare sunify.db to an application's file.db file. Unify DataServer/ELS uses sunify.db to create your application data dictionary, unify.db.

The application data dictionary

When you create a database, you are prompted with the following:

Do you wish to create a new data dictionary?

If you answer yes, sunify.db is used to create unify.db, your application data dictionary file. Unify DataServer/ELS stores schema design information about your application in unify.db. The schema design information stored in unify.db includes names for screens, menus, programs, tables, and fields.

The application data dictionary, unify.db, however, is confined by the limits set in the master data dictionary, sunify.db. For example, if sunify.db limits the total amount of fields allowed in your application to 800, this may not be enough fields. The ability to

customize the master data dictionary gives you access to sunify.db so you can increase the size of the application unify.db.

Customizing the data dictionary

The following describes each of the tables contained in the master data dictionary, followed by step-by-step master data dictionary modification and installation instructions. Be sure you understand the consequences of what you are changing before you modify the master data dictionary. Otherwise, disastrous effects might ensue!

The master data dictionary tables

The master data dictionary, sunify.db, contains the following tables:

dbnode	coreld
mprec	mline
groups	employ
empacc	grpacc
screen	scfield
mpscrf	fmtprog
record	field
fldrel	scffld
volume	fldsec
btree	btfld

You can change only the Expected number of records and the table descriptions.

NOTE: The tables dbnode, btree and volume are for internal use only, and the Expected Number of Records should not be changed. The volume and btree tables are limited in size by Unify DataServer/ELS to 8 and 155, respectively.

Each table description also contains the default Expected number of records. This is the number of records set in the master data dictionary when Unify DataServer/ELS is first installed.

dbnode	Database node record. dbnode is reserved for internal use only. The Expected number of records is 1.
coreld	The individual core load. coreld contains the names of UNIFY and HLI programs stored on disk that are used by the Menu Handler. The HLI program names are entered using execmnt. The default Expected number of records is 100.
mprec	Menu and program names. mprec contains menu names and the actual names used to invoke an HLI program entered through execmnt. The disk file names are contained in coreld. Execmnt updates this table. The default Expected number of records is 200.
mline	Menu line items. mline contains information about menu line items. The default Expected number of records is 300.
groups	Security groups. groups contains information about each security group. The default Expected number of records is 20.
employ	Employee (Users). employ contains information about each user entry. The default Expected number of records is 50.
empacc	Employee (user) access. empacc: contains permission information for users that are not defined in the groups table. Each access information entry corresponds to 1 program per employee. The default Expected number of records is 500.
grpacc	Group access. grpacc contains group access information for programs or screens. Each record corresponds to 1 program or screen per group ID. The default Expected number of records is 1000.
screen	Screen definitions. screen contains screen definition information. Each record corresponds to 1 screen. The default Expected number of records is 100.
scfield	Screen fields. scfield field information for each screen. Each entry corresponds to 1 screen field. The default Expected number of records is 3000.

mpscrf	Program screen intersections. mpscrf describes which screens are used for ENTER and/or SQL. Each record corresponds to an ENTER/SQL screen entry. The default Expected number of records is 100.
fmtprog	Formatting programs. fmtprog contains the formatting programs used for ENTER and SQL screens. 1 screen can correspond to a maximum of 8 fmtprog records. The default Expected number of records is 400.
record	Record types (tables). record contains information about the structure of a database table. Each entry corresponds to one table. The default Expected number of records is 152; the maximum allowed value is 252.
field	Field types. field contains information about the structure of a field. Each entry corresponds to an individual field. The default Expected number of records is 300.
fldrel	Reference intersections. fldrel contains field reference information. Each entry corresponds to 1 field reference. The default Expected number of records is 400.
scffld	Screen field to field intersections. scffld contains the database name of a screen field. That is, scffld contains the database field name you enter in PAINT when defining a screen field. The default Expected number of records is 1000.
volume	Database volumes. volume is reserved for internal use only. The Expected number of records is 8.
fldsec	Field level security. fldsec contains field password information. The default Expected number of records is 300.
btree	B-tree structures. btree is reserved for internal use only. The Expected number of records is 155.
btfld	B-tree fields. btfld contains B-tree field information. The default Expected number of records is 500.

The following procedure allows you to increase the Expected number of records in the master data dictionary.

A data dictionary table's data space is allocated in segments. Each table has a maximum of 10 data segments. The size of a segment depends upon the number of table records stored in the segment.

Unify DataServer/ELS uses the Expected number of records parameter to calculate the number of records per data segment. Unify DataServer/ELS uses the formula shown in Figure C.1 to determine the data segment size for a table.

$$\frac{\text{Expected Number of Records}}{6} = \text{Number of Records per Data Segment}$$

Figure C.1 Data segment size formula

The formula in Figure C.1 ensures that the Expected number of records requires only 6 of the 10 data segments. The remaining 4 segments are available as overflow storage.

For example, if you enter 600 as the Expected number of records for a table, the number of records per data segment would be 100:

$$\frac{600}{6} = 100 \text{ Records per Data Segment}$$

According to the formula in Figure C.1, if there are 100 records stored in each data segment, the 600 Expected Number of Records fit into 6 data segments. Because Unify DataServer/ELS allocates 10 segments automatically, there are still 4 segments (of 100 records each) available for table storage. These 4 segments provide storage space for an additional 400 records, for a total of 1000 table records in the 10 data segments.

The Expected number of records formula in Figure C.1 is important to consider if you want to change the Expected number of records for a particular database table. You must decide whether you want to change the estimated number of records or the actual number of records available for record storage.

If you want to change the estimated number of records, enter the new estimate as the Expected number of records parameter. The amount of actual table storage available will always be 30% greater than the estimate you entered. The amount overflow is determined by the expected number of records formula shown in Figure C.1.

If you want to allocate space for an actual number of records in a table, decrease the Expected Number of Records parameter by 30%. Decreasing the expected number of records by 30% compensates for the 30% overflow space that is automatically created using the formula in Figure C.1. Therefore, to allocate space for a specific number of records, enter an Expected Number of Records value which is 60% of the actual number of records you want to allocate space for.

The hash table size is based on the Expected number of records. Unify DataServer/ELS 's search performance is optimized when the hash table is no more than half full (hash table size is twice the Expected number of records entered). Therefore, changing the Expected number of records to reflect a specific number of records can impact performance when hash table entries exceed 50% of total hash table size.

Using the previous example where the Expected number of records is 600, if you wanted your application database to have storage space for only 600 records, you would use the equation:

```
60% of 600 Actual Records = 360 New Expected Number of Records
```

You would then enter 360 as the Expected number of records. Using the formula in Figure C.1, you would calculate the number of records per data segment as follows:

$$\frac{360}{6} = 60 \text{ records per data segment}$$

This allocates 6 segments of 60 records each (or 360 records), plus 4 more segments of 60 records each (or 240 records), totaling 600 actual records.

NOTE If the nearest prime number of half of the Expected number of records is less than 100, then Unify DataServer/ELS sets the hash table size to 100.

The following are steps for reconfiguring the master data dictionary. After this procedure, the master and application data dictionaries will be in sync.

Step1 Backup the Unify DataServer/ELS lib directory and the application database(s) you want to convert.

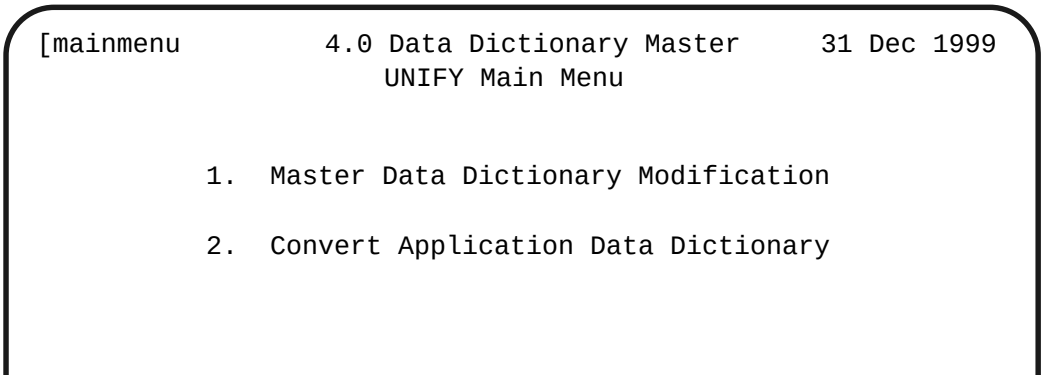
Step 2 Set the environment variable PATH, to the Unify DataServer/ELS bin directory.

Step 3 Set the environment variable Unify DataServer/ELS , to the Unify DataServer/ELS lib directory.

Step 4 Start the Reconfigure data dictionary option.

Type recondd.

This brings up the following menu:



Step 5 Select the Master data dictionary modification option. This option creates a temporary work directory, $UNIFY(ddwork and displays the following menu .

```
[mainmenu          4.0 Data Dictionary Master     31 Dec 1999
                       UNIFY Main Menu


              1.  Master Data Dictionary Modification

              2.  Convert Application Data Dictionary

              3.  Build Master Data Dictionary

              4.  Install Master Data Dictionary
```

Step 6 To change any of the system screen prompts, select the Modify system screens option. This is very useful for applications displaying screens in non-English languages. Normally, you do not need to change the system screen prompts.

!WARNING Change only screen field prompts. Never move screen fields on the screen. If you move screen fields you will no longer be able to use your screen.

Select the Customize Master Data Dictionary option to begin customizing the master data dictionary. The following menu displays: .

```
[mainmenu          4.0 Data Dictionary Master     31 Dec 1999
                        UNIFY Main Menu


LN  CMD  TABLE    EXPECTED  LONG NAME        DESCRIPTION
1        dbnode   1         data_base_node   node record
2        coreld   100       coreload         individual core load
3        mprec    200       menu_program     menu or program record
4        mline    300       menu_line        menu line item
5        groups   20        employee_group   security group
6        employ   50        employee         employee records
7        empacc   500       employee_access  group access record
8        grpacc   1000      group_access     group access record
9        screen   100       screen           screen definitions
10       scfield  3000      screen_field     screen filed definition
11       mpscrf   100       program_screen   program screen intersect
```

Step 7 Using the Data dictionary table section as a guide, you can make changes to the EXPECTED and the DESCRIPTION columns. This screen works the same as Modify database design, schent.

!WARNING Do not decrease the size of the application data dictionary, unify.db, for an existing application. If you decease the expected number below the actual number of entries in unify.db, any entry above the new expected number is destroyed. Therefore, only decrease the Expected number of records in the data dictionary before you actually start creating a database design for your application.

- ❖ When you have finished making the necessary changes, CTRL U to exit the Customize master data dictionary menu.

Step 8 Select the Build master data dictionary option in the Master data dictionary modification menu (dbdesign). This takes the changes you

made to the master data dictionary, and builds a new master data dictionary in the temporary work directory.

- ❖ Answer Y to the Proceed? prompt.

Step 9 When the master data dictionary has been built successfully, select the Install Master data dictionary option from the dbdesign menu. This step moves the new master data dictionary created in the temporary directory, and replaces the original masterdata dictionary in the Unify DataServer/ELS lib directory.

- ❖ Answer Y to the Proceed? prompt:

  No one should be using the database while installing a new master data dictionary in xxxxx. Proceed? (y/n)

Step 10 Once the master data dictionary has been installed successfully, return to the Unify DataServer/ELS Main Menu, mainmenu.

- ❖ CTRL P or mainmenu

Step 11 Select the Convert application data dictionary option to convert your application data dictionary (unify.db) using the new master data dictionary information.

Enter the directory location of the data dictionary you want to convert, or press RETURN, to the following prompt:

DBPATH is set to xxxxx
Enter new DBPATH or <cr> to continue

- ❖ Answer Y to the following prompt:

  No one should be using the database in xxxxx while installing a new data dictionary. Proceed? (y/n)

  NOTE: If you have multiple databases you want to convert, repeat this step for each application data base.

Step 12 Once you have successfully converted all your application data dictionaries, You are done. Exit to the Unify DataServer/ELS Main Menu.

- CTRL D

Appendix D: Unify DataServer/ELS messages

Unify DataServer/ELS has many useful messages. Usually they indicate errors, but often they display information about the current option. Many Unify DataServer/ELS messages have a second help level. You can get this additional help by responding to the message with ?

>>> <EMPTY>

Explanation: The Report option's output holding file has no data.

>>> A database design has not been entered.

Explanation: This message displays if the program cannot find a database design to compile.

Correction: Make sure that your environment variables or your current directory refer to the correct database and that a database design has been defined.

>>> A database field name was not entered. Try again?

Explanation: You cannot skip the FIELD NAME prompt.

Correction: Enter a y to return to the database field entry area and enter a valid field name; otherwise, the line will be discarded.

>>> A field name cannot be given twice in one B-tree definition.

Explanation: You can use a field name only once in each B-tree definition.

>>> A 'having' clause cannot contain nested aggregate functions.

Explanation: The having clause can only use simple aggregate functions. Nested aggregates are not recognized in this context.

Invalid Query:
```
select dept_no
from emp
group by dept_no
having avg(sum(salary)) >50000/
```

Correction:
```
select dept_no
from emp
group by dept_no
having sum(salary) > 50000/
```

>>> A 'having' clause must be preceded by a 'group by' clause.

Explanation: The having clause calculates the values of aggregate functions based on the sets of records specified by the group by clause. Therefore, a group by is required to use a having clause.

Invalid Query:
```
select job
from emp
having avg(salary) > l000/
```

Correction:
```
select job
from emp
group by job
having avg(salary) > 10000/
```

>>> A 'having' clause requires an aggregate function.

Explanation: The having clause is used to select records based on an aggregate property of a group. Consequently, you must use an aggregate function.

Invalid Query:
```
select job, salary
from emp
group by job
having salary > 1500/
```

Correction:
```
select job, avg(salary)
from emp
group by job
having avg(salary) > 1500/
```

>>> A line containing screen fields cannot be deleted

Explanation: The current line contains at least one screen field and cannot be deleted.

Correction: To delete a line containing screen fields, delete the fields, then delete the line.

>>> A line number must be entered

Explanation: When you modify or add menu lines, you must enter line numbers to indicate in what order the options should display.

Correction: Enter a number from one to 16.

>>> A literal tuple contains the wrong number of items.

Explanation: Every literal tuple must contain the same number of items as listed in the field specification list. Otherwise, SQL cannot determine what to do with the extra value or missing field.

Invalid Query:
```
select name
from emp
where <job, salary> = <'clerk*'>/
```

Correction:
```
select name
from emp
where <job, salary> = <'clerk*', 950>/
```

>>> A newly created field cannot be indexed before running one of these programs: Create Database, or Reconfigure Database.

Explanation: You are trying to create an index definition using a field that has not yet been compiled in the data dictionary.

Correction: You must run the appropriate option first. REUTRN gets you back to the FIELD NAME prompt to enter another field name.

>>> A screen field already exists with that name

Explanation: PAINT does not allow duplicate screen field names.

Correction: Choose another name.

>>> A sort command is not allowed on presorted input

Explanation: When the sorted option is used with the input command, RPT doesn't allow the sort command to be used.

>>> A string constant was expected.

Explanation: The syntax for the command requires a constant

NOTE: String constants must be enclosed in single quotes (')

Invalid Script: `separator`

Correction: separator

>>> A syntax error occurred near the string <xxxx>.

Explanation: This message helps you debug syntax errors. As the compiler was processing the statement on the specified line, an error was detected in the area shown by the string xxxx.

>>> A syntax error occurred at or near the end of the line.

Explanation: This message helps you debug syntax errors. As the compiler was processing the statement on the specified line, an error was detected near the end of the line.

>>> Access privilege does not allow deletion

Explanation: You must have delete privilege to remove a record from the database.

>>> Adding a line would push a screen field off the bottom

Explanation: The screen form design does not have enough space to add another line.

Correction: Try rearranging the screen form to create more space.

>>> Aggregate function imbalance in the 'select' clause.

Explanation: All aggregate functions in a single query must be nested to the same level. If you want to display both answers, you must perform two queries.

Invalid Query:
```
select avg(salary), sum(avg(salary))
from emp/
Correction:select avg(salary)
from emp/
```

or

```
select sum(avg(salary))
from emp
group by dept_no/
```

>>> Aggregate functions are not allowed in the 'where' clause.

Explanation: Aggregate functions can only be used in the having and select clauses. You must rewrite the query to use a having clause if you want to select records based on an aggregate function.

Invalid Query:
```
select job
from emp
where avg(salary) > 1000/
```

Correction:
```
select job
from emp
group by job
having avg(salary) > 1000/
```

>>> Aggregate functions may not be nested without a 'group by' clause.

Explanation: Because an aggregate function calculates a single value for a group of values, SQL does not let you apply an aggregate to a single value. This occurs when you try to nest aggregate functions without using a group by clause.

Invalid Query:
```
select sum(avg(salary))
from emp/
```

Correction:
```
select sum(avg(salary))
from emp
group by job/
```

>>> Aggregate functions nested too deeply in the 'select' clause.

Explanation: Aggregate functions can only be nested to two levels.

Invalid Query:
```
select max(sum(avg(salary)))
from emp/
```

Correction:
```
select max(sum(salary))
from emp
group by dept_no/
```

>>> An aggregate function is required when using a 'group by' clause.

Explanation: A group by clause can only be used to group records for the subsequent evaluation of an aggregate function. SQL does not let you group records and not calculate an aggregate function.

Invalid Query:
```
select job, salary
from emp
group by job/
```

Correction:
```
select job, min(salary)
from emp
group by job/
```

>>> An ascending /descending code was not entered. Try again?

Explanation: Every index key field must be specified assorted in ascending or descending order.

Correction: Enter an a for ascending or d for descending sort. A no answer at this prompt discards the current line.

>>> An error occurred while writing the compiled output file

Explanation: This is an internal system error.

Correction: Contact Unify Corporation Customer Support.

>>> An error was encountered while writing the B-tree index file.

Explanation: This write error can occur when you add or rebuild an index.

Correction: Before you add or rebuild another index, make sure the database file system has enough space.

>>> An expression cannot be compared with a literal tuple list.

Explanation: An expression yields a single value, but a literal tuple contains several values. SQL cannot determine what to do with the extra values in the comparison.Invalid

Query:
```
select name
from emp
where salary+commission is in
<2500, 'salesman*'>,
<800, 'clerk*' >,
<1100, 'programmer*'>)/
```

Correction:
```
select name
from emp
where salary+commission is in
<2500, 800, 1100>/
```

>>> An illegal literal was used for the default clause

Explanation: A constant in the default clause either doesn't match the current field's data type or isn't a valid value for the data type.

Correction: Check the data type of the field named in the field statement, and check the constant for a possible mistake in entry.

>>> An integer constant was expected.

Explanation: The syntax for the command requires an integer.

Invalid Script:
```
bottom margin 7e
```

Correction:
```
bottom margin 7
```

>>> An internal system error has occurred.

Explanation: Read Database Backup or Write Database Backup has encountered an internal error it cannot work around.

Correction: For more information about the error, see the errlog file. Correct the error and try again.

>>> An Internal system error has occurred. Try creating the database, screens and menus manually

Explanation: An unexpected error has occurred.

Correction: Try creating the database, screen forms, and the menus with the appropriate options.

>>> Are you sure you want to abort?

Explanation: You have requested that the Read Database Backup or Write Database Backup procedure stop. You must confirm the request.

For example, this message displays when you answer N to the "Please mount backup media volume nnn" message.

Correction: If you answer Y or y to confirm, the procedure stops. If you answer N or n, the procedure continues.

>>> At least one index was not rebuilt.

Explanation: This message displays if the Rebuild B-tree Indexes program cannot rebuild all the indexes. Detailed error messages are sent to the file called errlog in your database application's bin directory.

Correction: Exit this program. Make sure the database file system has enough space. Fix the problems described in the error log, then try to rebuild the indexes.

>>> B-tree index xxxx is missing or has incorrect permissions. Rebuild it if you want it to be used.

Explanation: This message displays if the B-tree index file has been removed or has incorrect access privileges.

Correction: If the B-tree index file exists, make sure you have read and write privileges for the file.

If the B-tree index file does not exist and you want to use this B-tree with your database, rebuild the B-tree index.

>>> Backup complete

Explanation: The backup has been successfully written. Any keyboard response returns you to the menu.

>>> Backup device is off-line. Retry?

Explanation: The Read Database Backup device is either not turned on, or is not on line.

Correction: If you want to try again, correct the problem, and respond with a Y or y. The backup volume will begin to read.

If you want to exit the program, respond with CTRL U, N or n.

>>> Backup device off line or not write enabled - Try again?

Explanation: This can be one of several problems: The Write Database Backup device is not turned on. It is not on-line. The diskette or tape is not write-enabled. Or, your operating system user-ID does not have read/write privileges for the backup device.

Correction: If you want to try again, fix the problem and answer y or Y. The backup volume will begin to write.

If you want to exit the program, answer n, N, or CTRL U.

>>> Backup media block size is nnn.

Explanation: Read Database Backup has determined that the backup media volumes were written with a block size of nnn. The program will use this block size to read the backup tapes or diskettes.

This message only displays if nnn *is* different from the block size specified in BUBLK.

>>> Backup media volume nnn was mounted.

Explanation: This message informs you that volume nnn is mounted, instead of the volume requested in the "Please mount backup media volume nnn" message.

Correction: Replace volume nnn with the correct volume.

>>> Bad media: unable to read/write header block.

Explanation: In trying to Read or Write Database Backup, the first read from or write to the backup tape or diskette failed.

Correction: For Read, this is probably not a backup volume.Make sure the tape or diskette is a backup volume, then retry.

For Write, this could be because the media is damaged. Replace the backup tape or diskette and try again.

>>> BINARY type fields are not allowed.

Explanation: Only NUMERIC, STRING, DATE, LDATE, TIME, FLOAT, or AMOUNT fields can be used in an index definition.

>>> Blocksize too big, decrease the value of the BUBLK environment variable

Explanation: There is no data block size to match the block size specified in *BUBLK*.

Correction: Decrease the size of the *BUBLK* environment variable to match the data block. For more information about setting environment variables, see Section 5.

>>> Blocksize too small, increase the value of the BUBLK environment variable.

Explanation: For Read Database Backup, the block size specified in *BUBLK* is smaller than the block size on the backup tape or diskette. For Write Database Backup, the block size specified in *BUBLK* is too small to store the data in.

Correction: Increase the size of the *BUBLK* environment variable to a larger value. For more information about setting environment variables, see Section 5.

>>> Cannot add characters to a line ending with a wrap around field

Explanation: Because of a wraparound field's characteristics, text cannot be added to the line if it shifts the wraparound field to the right.

Correction: To add characters, delete the wraparound field, add the characters.' and reenter the field.

>>> Cannot change video mode of a field

Explanation: Screen fields can only be displayed in normal video mode.

>>> Cannot change video mode on more than one line at a time

Explanation: PAINT can only change video mode for one line at a time.

>>> Cannot delete characters on a line ending with a wrap around field

Explanation: Because of a wraparound field's characteristics, text cannot be deleted if it shifts the wraparound field to the left.

Correction: To delete characters, delete the wraparound field, delete the characters, and reenter the field.

>>> Cannot delete combination

Explanation**:** A COMB field cannot be deleted until after its corresponding component fields have been deleted.

>>> Cannot find or cannot display the help file

Explanation: The help file for the current field cannot be found or cannot be displayed.

Correction: Make sure the help file name is spelled correctly in the Advanced Field Attributes file.

Make sure you have spelled the file name correctly and that the file is in the help documentation directory. Help documentation should be in ../hdoc.

Make sure the help file has proper access permissions set for it to be read by ENTER.

>>> Cannot insert in the middle of a field

Explanation: A field is a predefined area. Screen text cannot be inserted in the middle of a field.

>>> Cannot open a line below screen's last line

Explanation: You are at the last line of the screen's text entry area. No more lines can be added.

>>> Cannot open a line in the middle of a wrap around field

Explanation: Wraparound fields must be displayed on consecutive lines. You cannot open a new line if it would split the wraparound field.

>>> Cannot open xxxx file for input.

Explanation: The compiler cannot open the field attributes file named xxxx.

Correction: Check the spelling of the name and make sure write privileges are set for the file.

>>> Cannot open the unify.afa file for output.

Explanation: The compiler cannot open or create the file unify.afa. This file stores the compiled field attribute specifications.

Correction: If the file exists, make sure you have write privileges for the directory and for the file unify.afa.

>>> Cannot transfer a line into the middle of a wrap around field

Explanation: Wraparound fields must be displayed on consecutive lines. You cannot transfer a line to where it would split a wraparound field.

>>> Cannot transfer a line starting or ending with a wrap around field

Explanation: Because of a wraparound field's characteristics, you cannot move a line containing a wraparound field.

Correction: To move the line, move each field separately, and rearrange the prompts; or delete the wraparound field(s), move the line, and reenter the field(s).

>>> Can't create data dictionary

Explanation: The data dictionary and database file cannot be recovered. The data dictionary and database are now invalid.

Correction: Any keyboard response returns you to the menu. To recover, you must either read the backup as a user who can create files in the database directory, or change the access mode of the directory so you can create files.

>>> Can't create 'xxxx'.

Explanation: SQL cannot create the ASCII or binary file named xxxx.

Correction: Check the read and write permissions of the current directory. Make sure the current user has write privileges for the directory.

>>> Can't open database file - BUDB

Explanation: The database file cannot be opened. Any keyboard response returns you to the menu.

Correction: Make sure you have read/write privileges for the database file.

>>> Can't open data dictionary

Explanation: The data dictionary cannot be opened. Any keyboard response returns you to the menu.

Correction: Make sure you have read/write privileges for the data dictionary file.

>>> Can't open specification file: xxxx.

Explanation: The specification file cannot be opened.

Correction: Make sure the file name is spelled correctly and you have read privileges for the file.

>>> Can't open volume xxxx - readtemp

Explanation: The database volume file specified in Define Database Volumes (volmnt) cannot be opened.

Correction: Check the file permissions of the volume file and confirm that the file exists.

>>> Changing this parameter will turn transaction logging off

Explanation: If transaction logging is turned ON, any changes to the transaction logging definition cause logging to be turned OFF until a backup is performed.

>>> Column number must be between 0 and 79

Explanation: On an 80-column terminal, only columns 0 through 79 are valid x coordinates for the go to command.

>>> Column number must be between 0 and 131

Explanation: On a 132-column terminal, only columns 0 through 131 are valid x coordinates for the go to command.

>>> COMB type fields are not allowed (give their elements separately).

Explanation: Only NUMERIC, STRING, DATE, LDATE, TIME, FLOAT, or AMOUNT fields can be used in an index definition.

>>> Compilation complete.

Explanation: The source file has been successfully compiled and the output placed in the file unify.afa in the directory set by *DBFATH.*

>>> Complete. Please enter RETURN to continue.

Explanation: This part of the Report Option process is finished.

>>> Database is empty, use Create Database

Explanation: There is no data in the database.

Correction: Use Create Database to create the database.

>>> Data dictionary not updated

Explanation: The temporary file used for reconfiguration could not be created for some reason, so the data dictionary was not updated.

Correction: If you are using diskettes or tape for the temporary file, check your backup device to be sure it is on line or write-enabled. Or, if you are using the disk as the temporary file, check that you have write permission on the current directory.

>>> DBLOAD: Unable to open xxxx.

Explanation: The ASCII input file cannot be opened for reading.

Correction: Check the spelling of the name and make sure you have read privileges for the file.

>>> DBLOAD: Insufficient memory.

Explanation: There is not enough memory for DBLOAD to finish initializing.

Correction: Try increasing the amount of memory allowed for initialization. This can require an operating system regeneration.

>>> Deadlock detected from system locking

Explanation: Either this record is being accessed at another terminal, or a lockfile may be present (LOCKd) in your $DBPATH directory.

Correction: If a LOCKd is present, you must remove it before you can continue. Be sure no one is using the file locking facility before you remove LOCKd.

>>> Display next page? [RETURN] continues, [n] terminates

Explanation: The output from the current Report Option process does not fit on one screen.

Correction: Press RETURN to display the next screen of data. Press n to end the report process and resume the program.

>>> Do not use 0 to invoke the paint command

Explanation: The 0 command key can only be used with the PAINT command co10 (go to column 0).

Correction: Change the PAINT command from 0 to another command key.

>>> Do not use the same key for different Menu Handier commands.

Explanation: A duplicate command key is used in the menu section of unicap. Note the command table is displayed, indicating the conflicting command names.

Correction: Try modifying the unicap file.

>>> Do not use the same key for different paint commands

Explanation: Each PAINT command definition must have unique command keys.

Correction: Modify your unicap file to eliminate any duplicate command keys in the PAINT section.

>>> Do you really want to exit, [Y]es or [N]o?

Explanation: You have entered the exit command. Leaving PAINT this way does not allow you to save the current screen form.

Correction: Answer y (yes) or n (no).

>>> Do you want to change the number of blocks per volume from nnn?

Explanation: If the number of blocks that can be written on a tape or diskette is less than specified in BUTAPESZ, this lets you change the effective value of the BUTAPESZ environment variable to the correct number of blocks.

Correction: If you enter Y or y to change the number of blocks per volume, Write Database Backup displays another prompt:

```
Change the number of blocks to 0 (whatever will fit)?
```

If you answer Y, the effective value of BUTAPESZ is changed to 0. If you answer Nand mmm is an acceptable value, Write Database Backup displays the following prompt:

```
Change the number of blocks to mmm?
```

If you answer Y to this prompt, the value of BUTAPESZ is changed to the number of blocks that actually fit on the backup volume.

>>> Do you want to just try the replay program again?

Explanation: An error occurred while replaying the transaction log. This can indicate that the transaction log is bad and a replay cannot be made.

Correction: If you want to try to rerun it without changes, respond with a Y or y.

>>> Do you want to read in a backup again?

Explanation: An error occurred while reading the database backup file.

Correction: If you want to try to read it in again, respond with Y or y.

>>> Duplicate xxxx statement.

Explanation: A duplicate xxxx (FYI, ERROR, or HELP) statement is in a single field section.

Correction: Remove one of the duplicate statements.

>>> Duplicate field statement for xxxx field name.

Explanation: The table section has two field statements for the same field name.

Correction: Remove one of the duplicate field statements.

>>> Duplicate value for B-tree nnn, record rrr.

Explanation: The table being loaded has a "no duplicates" B-tree index, AND record number rrr in the input file contains a duplicate value for this B-tree

index. DBLOAD neither adds the record if it is new, nor updates the field if the record exists.

>>> Edit command too long

Explanation: The complete edit command line is limited to 130 characters. The command line is the sum of the current editor's name length plus the length of the file name argument.

>>> Either 'desc' or a named expression was expected

Explanation: Improper syntax was used in a sort command or in one of its expressions. You may have included a comma at the end of the sort command line.

Invalid Script: `sort zip_code, <a1+a2> 0_num - d_num,`

Correction: `sort zip_pode. <new_num> 0_num - d_num`

>>> Enter e, c, or p

Explanation: The only valid responses to the operation prompt are e for edit, c for compile, and p for print.

Correction: Re-enter the appropriate command.

>>> Environment variable DBNAME: xxxx is an illegal value

Explanation: This message displays if the environment variable DBNAME is set to an illegal value.

Correction: Make sure DBNAME is set to a valid name.

>>> Error from B-tree nnn.
Status is ddd.
This B-tree should be rebuilt after the load.

Explanation: A problem has occurred while updating B-tree index nnn.

Correction: Rebuild this B-tree index after the database load.

>>> Error from B-tree back out.
B-tree nnn, Status xxx, Record rrr.
This B-tree should be rebuilt after the load.

Explanation: While recovering from a previous error, DBLOAD found another error. This B-tree index can be invalid.

Correction: Rebuild the B-tree index after the database load.

>>> Errors encountered. Press RETURN to continue.

Explanation: Some errors were detected while compiling the Advanced Field Attributes. All error messages were sent to the error file, fields.afa.err.

Correction: Press RETURN for the editor to display the error file.

If you are using vi, you can enter :n to switch to the source file. After that, you can enter :e# to switch between the two files.

>>> Fatal error

Explanation: SQL or RPT has encountered an internal error that it cannot work around. This error message is preceded by an error message indicating the type of error.

Correction: Try rewriting the query or script differently. If the problem persists, report the problem to Unify Corporation.

>>> Field: ffff requires a password, use 'unlock.'
Field: xxxx.ffff requires a password, use 'unlock.'

Explanation: The indicated field in the select, where, or having statement has been protected with Field Level Security.

Correction: You must unlock the field before you can list it.

>>> Field exists

Explanation: A field with the name you are trying to enter already exists in the database.

Correction: Check the spelling or use a different field name.

>>> Field ffff is not in table xxxx

Explanation: The field named ffff exists in the database, but is not in the table xxxx.

Correction: Check the spelling of the field or table name.

>>> Field is referenced

Explanation: You cannot delete a field referenced by another field in the database.

Correction: You must first remove any reference to this field before it can be deleted.

>>> Field limit exceeded

Explanation: This table already has 256 fields; no more can be added. This is an internal system error.

>>> Field limit will be exceeded

Explanation: If you add another field to the table, the field limit of 256 fields will be exceeded. This field cannot be added.

>>> Field must be a parent

Explanation: The field name entered as the combination field entry for the current field is not of the data type COMB.

Correction: To make this field part of a COMB field, you must specify a COMB field name in the current table in the COMB. FIELD column.

>>> Field must reference a key

Explanation: You are trying to use a non-key field as a reference field. Only the primary key of another table can be used as a reference field.

Correction: Check the entry for misspellings.

>>> Field must reference another table

Explanation: You are trying to use a field in the current table as a reference field. Only the primary key of another table can be used as a reference field.

Correction: Check the entry for misspellings.

>>> Field unknown

Explanation: The field name that is being entered as a reference field or combination field cannot be found.

Correction: Check the spelling of the field name to be entered.

>>> Fields can't have table names

Explanation: The name you are trying to enter as a field name is already used as a table name.

Correction: Use a different field name.

>>> Illegal field type

Explanation: Fields must be of type NUMERIC, AMOUNT, FLOAT, DATE, LDATE, TIME, STRING, TEXT, or BINARY.

>>> Illegal field type (COMB)

Explanation: COMB field types cannot be used directly.

Correction: Enter each component of the COMB field separately.

>>> Illegal use of variable: xxxx

Explanation: A variable cannot be used in this context. This error can be the result of a misspelled function, keyword, variable, or field name.

Invalid Script:
```
sort czip, x
.
.
.
before report
set x to 0
```

Correction:
```
sort czip, x-amount
.
.
.
before report
set x to 0
```

>>> Input field xxxx.ffff does not have its type specified, and it is not a Unify DataServer/ELS field.

Explanation: RPT cannot find field ffff in the database table xxxx. If this input field is an arbitrary name, there may be no data type and length specification.

When using the syntax table.fieldname, make sure the input field name, ffff, is the access, or long, name for the database field.

CAUTION: This message is also generated when the report field is matched to a COMB database field.

Invalid Script:
```
input
cnum,
cname
detail
print cnum, cname
end
```

Correction:
```
input
customer.enum,
customer.cname
```

```
detail
print cnum, cname
end
```

or

```
input
cnum [numeric 5],
cname [string 30]
detail
print cnum, cname
end
```

>>> Insufficient memory for arglist

Explanation: When describing a user function call in a script, too many arguments were used to call the function.

Correction: Try using fewer arguments.

>>> Insufficient memory for backup device structures.

Explanation: There is not enough memory left for the backup device internal data structures. The value specified in BUBLK may be too large.

>>> Insufficient memory for logical transaction table

Explanation: A fatal error has occurred, causing the replay program to fail, because the program cannot allocate enough memory to hold the incomplete logical transaction table.

This error should only occur if you have a large number of incomplete logical transaction groups, and your hardware has limited address space.

>>> Insufficient memory for sort line.

Explanation: Either the group by or the order by clause has too many elements.

Correction: Use fewer sort elements.

>>> Insufficient memory for successful compile.

Explanation: This is an internal error. Your system does not have enough memory to complete the compile process.

Correction: Try reducing the complexity of the Advanced Field Attributes file.

>>> Insufficient memory for the record relocation table

Explanation: A fatal error has occurred, causing the replay the program to fail, because the program cannot allocate enough memory to hold the record relocation table.

The record relocation table keeps track of record locations that have changed because of incomplete transactions.

This error should only occur if you have a large number of errors or incomplete logical transaction groups, and your hardware has limited address space.

>>> Internal error In ckset nnn III

Explanation: You are trying to set a variable to an unrecognized type.

Invalid Script:
```
input
i_time [time],
value [numeric 4]
before report
set x to DATA
.
.
.
detail
print x, i_time, value
```

Correction:
```
input
i_time [time],
value [numeric 4]
before report
set x to 'DATA'
```

```
detail
print x, i_time, value
```

>>> Internal system error, unable to execute the backup driver.

Explanation: Read Database Backup or Write Database Backup cannot find or cannot run the backup device driver.

The backup device driver is either the default, BUDRVR, or a custom version of BUDRVR specified in the BUDRVR environment variable.

Correction: Make sure BUDRVR, or the custom version of BUDRVR specified in BUDRVR, is an executable file. Also make sure the file is in a directory specified in your PATH environment variable.

>>> Invalid column number nnn generated by print statement on line ddd

Explanation: This message displays at run time. It only identifies the approximate location of the print statement. The column number nnn is greater than the set page width.

The actual statement can occur before the line number indicated. Note that nnn is one less than the actual number shown in the script.

Invalid Script:

```
.
.
.
width 80
.
.
.
print 'New State' in column 89
```

Correction:

```
.
.
.
width 80
print 'New State' in column 65
```

>>> Invalid date.

Explanation: The format of an Idate constant is incorrect. The default format for an Idate is MMIDDIYY.

If the LDATEFMT environment variable is set, the specified format is used for Idate constants. See Section 5, for more information about setting the LDATEFMT environment variable.

Invalid Script:
```
set start_date to 12/35/84
```

Correction:
```
set start_date to 12/31/84
```

>>> Invalid field: ffff

Explanation: SQL cannot recognize the field name ffff. Make sure you are using the correct long name or access name for the field.

Invalid Query:
```
select ename from emp/
```

Correction:
```
select name from emp/
```

>>> Invalid field length

Explanation: You have entered an invalid field length for the associated field type.

Correction: Refer to Appendix A or Database Fields in Section 8.1 for valid field lengths.

>>> Invalid field name

Explanation: You have entered a database field name that cannot be found in the data dictionary.

Correction: Check the spelling of the field name.

>>> Invalid field specification: xxxx.ffff

Explanation: SQL cannot recognize the indicated field specification, where xxxx is the table, and ffff is the field. In the following invalid query, the table name is misspelled.

Invalid Query:
```
select epm.name from emp/
```

Correction:
```
select emp.name from emp/
```

>>> Invalid format for field ffff, record nnn

Explanation: The field named ffff in record number nnn of the input file has an invalid format.

Correction: Look for an invalid date, time, or number with embedded alpha characters. If every record has the same problem, your specification file might not match the input file.

>>> Invalid 'group by' clause.

Explanation: The syntax of the group by clause is incorrect. This can be caused by various errors. The problem in the following invalid query is that it tries to group by computed fields. 4

Invalid Query:
```
select avg(salary)
from emp
group by (salary+commission)/
```

Correction:
```
select avg(salary)
from emp
group by salary, commission/
```

>>> Invalid 'having' clause.

Explanation: The syntax of the having clause is incorrect. This can be caused by various errors. The problem in the following invalid query is that the having clause needs a Boolean expression, using a comparison operator such as <, or >.

Invalid Query:
```
select dept_no
from emp
group by dept_no
having max(avg(salary))/
```

Correction:
```
select dept_no
from emp
group by
dept_no
having avg(salary) = select
max(avg(salary))
from emp
group by dept_no/
```

>>> Invalid help selection

Explanation: You are trying to select help by menu line number, as in help nnn, where nnn is not a valid selection for this menu.

>>> Invalid length

Explanation: The value entered for this field's length is invalid. The maximum display lengths for the various field types are as follows:

NUMERIC	- 9
FLOAT	- -179
AMOUNT	- 11
DATE	- -n/a, defaults to 8
LDATE	- -n/a, defaults to l l
TIME	- -n/a, defaults to 5
STRING	- -256
TEXT	- -256
BINARY	- -n/a
COMB	- -n/a, computed by system

>>> Invalid query block.

Explanation: SQL cannot recognize the query expression. This can be caused by various errors, such as a character or space after the final slash (/). The problem in the following invalid query is that the keyword select is misspelled.

Invalid Query: `sellect name from emp/`

Correction: `select name from emp/`

>>> Invalid range for field ffff, record nnn.

Explanation: The value entered for field ffff in record number nnn of the input file is not a legal value. DBLOAD has not updated this record.

Correction: Check the value for mistakes. Set legal values with the Advanced Field Attributes feature.

>>> Invalid record location specification, record nnn.

Explanation: The record location for record number nnn of the input file cannot be understood. DBLOAD has not updated this record.

>>> Invalid response: enter 'device' or 'file'

Explanation: The only two valid transaction log types are device and file.

>>> Invalid response: enter 'yes' or 'no'

Explanation: The only valid responses to the prompt are Y (Yes) and N (No).

>>> Invalid 'select' list syntax.

Explanation: The syntax of the select clause is incorrect. This can be caused by various errors. In the following invalid query, the fields to be selected are enclosed in angle brackets (<>). The angle brackets cannot be used this way.

Invalid, Query:
```
select <name, job>
from emp/
```

Correction:
```
select name, job from emp/
```

>>> Invalid selection

Explanation: You are trying to select an option that cannot be found in the data dictionary, is not on the menu, or that you are not allowed to execute.

>>> Invalid status: enter 'on' or 'off'

Explanation: The only valid responses to the prompt are on and off.

>>> Invalid table: xxxx

Explanation: The table named xxxx doesn't exist in the database you are trying to load.

Correction: Make sure you are using the right database file and the right table, and that the file name and field name are spelled correctly.

>>> Invalid table: xxxx

Explanation: SQL cannot recognize the table name xxxx used in the query. The problem in the following invalid query is that the table name is misspelled.

Invalid Query:
```
select name from epm/
```

Correction:
```
select name from emp/
```

>>> Invalid table specification in the 'select' clause: xxxx

Explanation: SQL cannot recognize the table name xxxx used before the.*. The problem in the following invalid query is that the table name is misspelled.

Invalid Query:
```
select epm.* from emp/
```

Correction: `select emp.* from emp/`

>>> Invalid time.

Explanation: The format of a time constant is incorrect. TIME constants are of the form HH:MM, where HH is the hour (0 to 23) and MM is the minute(0 to 59). For example, the following invalid RPT script tries to set a time variable to an incorrect format.

Invalid Script: `set report_time to 12:344`

Correction: `set report_time to 12:34`

>>> Invalid type

Explanation: The field type just entered is not valid. The valid field types are NUMERIC, FLOAT, STRING, DATE, TIME, AMOUNT, COMB, BINARY, LDATE, and TEXT.

>>> Invalid Type Entered

Explanation: This message displays if you enter something other than F, f, D, or d as the volume type in Define Database Volumes.

Correction: Re-enter the volume type with a valid value.

>>> Invalid usage, use: afac srcfile [errfile]

Explanation: The compile command you have entered is wrong.

Correction: Enter the compiler name afac, followed by the name of the Advanced Field Attributes file, srcfile. If you don't want error messages sent to the standard error output, you can specify a file to receive error messages, errfile.

>>> Invalid value for field, ffff, record nnn

Explanation: The field named ffff has an explicit relationship with another field in the database, AND the value for ffff in record number nnn of the input file doesn't match its reference file.

Correction: Either insert a record in the reference file that matches the value of the field, or change the value so it matches an existing record.

>>> Invalid 'where' clause.

Explantation: The syntax of the where clause is incorrect. This can be caused by various errors. The problem in the following invalid query is that the clause lacks the Boolean expression needed to perform a comparison.

Invalid Query:
```
select name
from emp
where salary/
```

Correction:
```
select name
from emp
where salary > commission/
```

>>> ffff is an invalid field name for table xxxx.

Explanation: The field name you specified in the unlock command is not in the table you specified.

Correction: Make sure the field name is spelled correctly.

>>> ffff is an invalid field name.

Explanation: A field name specified in the field list for the insert statement is incorrect or has been misspelled.

Correction: Make sure the field name is spelled correctly.

>>> xxxx is an invalid file name.

Explanation: The ASCII or binary file named xxxx does not exist or cannot be read.

Correction: Make sure you have spelled the name correctly and you have access privileges to read the file.

>>> pppp is an invalid Password for xxxx.ffff.

Explanation: The password you specified to unlock the field is incorrect.

Correction: Make sure you have spelled the password correctly.

>>> xxxx is an invalid table.

Explanation: The indicated table does not exist in the database.

Correction: Make sure you have spelled the name correctly.

>>> xxxx is an unrecognized function name.

Explanation: The function name does not exist or cannot be read. You may have misspelled the function.

>>> <ffff> is not a field of <xxxx> table.

Explanation: The field named ffff in the specified field statement is not a field in table xxxx.

Correction: Check for a possible misspelling of the field name, and make sure this field statement is within the correct table section.

Left end of normal area is not on left edge of a video field

Explanation: You are trying to change a video field back to normal mode, but the left end of the normal area does not match the left edge of the existing video field.

Correction: When changing video modes, you must change the complete video field.

>>> Left type is ////, right type is rrrr, operator is xxxx.

Explanation: The data types of the two operands in the expression do not agree and cannot be converted. You may have used the wrong field or variable name.

Invalid Script:
```
input
cnum [numeric 3],
name [string 30],
zip [string 5],
.
.
.
if zip = 95648 then
.
.
.
```

Correction:
```
input
cnum [numeric 3],
name [string 30],
zip (string 51.
.
.
.
if zip = '95648' then
.
.
.
```

>>> Line must be 1-16

Explanation: Menus can only have 16 menu lines per menu.

Correction: Enter a number from one to 16.

>>> Line number must be between 3 and 20

Explanation: Only line numbers 3 through 20 are valid y coordinates for the go to command.

>>> Load interrupted. nnn records added, mmm records updated.

Explanation: The data load process has been interrupted. By termination, DBLOAD has added nnn records and updated mmm records.

>>> Media too small: unable to write data blocks.

Explanation: Only the backup header block fits on the tape or diskette

>>> Menu or Prog already exists

Explanation: You are trying to add a menu line for a menu or program that is on this menu.

Correction: List menus and options only once on a menu.

>>> Menu unknown

Explanation: The menu you are trying to access cannot be found.

Correction: If you want to add this menu, make sure you are in add mode. If you want to modify or delete this menu, make sure you spell the menu name correctly.

You can use Print Menus to get a list of menus. For more information, see Section 11.2.

>>> Message statement not preceded by a field statement.

Explanation: No current field statement precedes the message statement on the indicated line. The field statement tells the system what field is associated with the message.

Correction: Precede all message statements with field statements.

>>> Mount NEW transaction log

Explanation: After you remove the old transaction log volume used for the replay program, mount a new diskette in the device to record new transactions.

>>> Mount the NEW transaction log.

Explanation: The backup is done. Remount the diskette or tape used to record transactions.

>>> Mount the transaction log

Explanation: If the transaction log file is a removable device, the program asks you to mount the current transaction log file diskette or tape before you start the replay program. The log must match the backup just read.

>>> Mounted backup media is not part of this backup.

Explanation: The mounted tape or diskette is a volume from a different backup.

Correction: Replace the mounted volume with the correct tape or diskette.

>>> Mounted media is not a backup volume.

Explanation: The tape or diskette in the backup device is not a backup volume.

Correction: Replace the tape or diskette with the correct volume and retry.

>>> nnn blocks on backup media volume mmm.

Explanation: This message reports the number of blocks read from or written to the backup tape or diskette. You can use this information to make sure the same number of blocks is read from the media as is written to it. This lets you watch for errors not detected by the program.

>>> Names must start with a letter and have only letters, digits and _'s

Explanation: You can only use alphabetic or numeric characters and underscores in Unify DataServer/ELS name entries. The name must also start with a letter.

Correction: Re-enter the name.

>>> No help documentation file is available

Explanation: PAINT cannot find the file paint.dflt in the help documentation directory.

Correction: Make sure you have read permission for the $UNIFY/hdoc directory, and make sure the file paint.dflt is present.

>>> No help is available on this subject.

Explanation: The help file for the keyword you specified does not exist or cannot be read.

Correction: Make sure you have correctly spelled the keyword you want help about. Otherwise, make sure the environment variable UNIFY is set correctly, so the help directory can be found.

>>> No more pages

Explanation: This is the last or only page of this table or field list. The next page command cannot work.

>>> No more records

Explanation: This is the only or last record selected. The selection file contains no more records.

>>> No previous page

Explanation: This is the first or only page of this table or field list. The previous page command cannot work.

>>> No previous record

Explanation: The previous record command cannot work because you are at the first record in the selection file.

>>> No space for video control spaces

Explanation: Some terminals require a blank character position to turn video control ON and OFF, PAINT does not allow these character positions to be shifted off the screen. Therefore, with the current screen form design, you cannot set this video mode.

Correction: Create more space around the area where you want to set video attributes.

>>> No table statement found for <ffff> field.

Explanation: The field ffff as defined in the field statement on the specified line does not have an associated table statement.

Correction: Place all field statements in a table section.

>>> Not enough memory

Explanation: The query is too long or is nested too deeply for the hardware you are using.

Correction: Simplify the query or divide it into smaller queries. You can use the results of each small query to form the next query. Otherwise, you can try adjusting the SQL internal table sizes to accommodate the query. For information on SQL internal table sizes and environment variables, refer to Section 16.3.

>>> Not enough memory, field not added

Explanation: This is an internal error. There is not enough memory to perform the indicated operation.

Correction: Reduce the number of screen fields or the amount of prompt text.

>>> Not enough memory, PAINT will terminate

Explanation: This is an internal error. There is not enough memory to perform the indicated operation.

Correction: Reduce the number of screen fields or the amount of prompt text.

>>> Not enough memory for all the fields in this screen

Explanation: This is an internal error. There is not enough memory to perform the indicated operation.

Correction: Reduce the number of screen fields or the amount of prompt text.

>>> Not enough memory to process (saved) screen

Explanation: This is an internal error. There is not enough memory to perform the indicated operation.

Correction: Reduce the number of screen fields or the amount of prompt text.

>>> xxxx not found

Explanation: You are trying to enter a value for a field, and no record in the parent (referenced) table, xxxx, has this value. The string xxxx is the long name of the table.

Correction: Only enter values for this field that match existing values for its referenced field because of the fields' explicit relationship.

>>> on or about line nnn

Explanation: This message is sent to the standard error output when a syntax error is detected. It indicates approximately where the syntax error occurs in the query or report script.

>>> Only nnn blocks read from backup media. Continue?

Explanation: This message displays when the program cannot read the number of blocks specified in BUTAPESZ.

If you enter Y or y to continue, Read Database Backup will prompt you to mount the next backup tape or diskette.

>>> Only nnn blocks written to backup media. Continue?

Explanation: This message displays when Write Database Backup is unable to read the number of blocks specified in BUTAPESZ.

Correction: To continue, enter Y or y, and the program will prompt you to mount the next backup volume.

>>> Only date field types are valid for the 'today' default.

Explanation: The keyword today cannot be used in this field statement because the current field is not of type DATE or LDATE.

>>> Only numeric field types, length 5-9 are valid for the 'unique' default.

Explanation: The keyword unique cannot be used in this field statement because the current field is not a NUMERIC field with a length of 5-9 characters.

>>> Only time field types are valid for the 'hour' default.

Explanation: The keyword hour cannot be used in this field statement because the current field is not of type TIME.

>>> Operation not allowed on a video control space

Explanation: Because of the terminal's characteristics the current position, although blank, contains a video control character. Therefore, PAINT cannot perform the requested operation in this space.

>>> Operator abort

Explanation: You have asked to stop the Read Database Backup or Write Database Backup program; as for example, when you answer N to a "Continue?" prompt.

>>> Other records reference this field. You cannot change its value now

Explanation: Other records in the database also reference this field. Therefore, this field's value cannot be changed at this time.

Correction: You must change the related records so they do not reference this record.

>>> Output width overflow

Explanation: This message displays at run time when a print line's output is longer than the page width. This message also displays when you include a no newline in your last print statement.

Invalid Script:
```
input
cnumber [number 8),
name [string 30],
state [string 2]
width 20
detail
print 'This is a listing of customers'
centered
skip 2
print cnumber, name, state in column 18
```

Correction:
```
input
cnumber [number 8],
name [string 30],
state (string 2]
width 35
detail
print 'This is a listing of customers'
centered
skip 2
```

```
print cnumber, state in column 3
print name
```

>>> Parent must be of the same table

Explanation: You are trying to use a field in another table as the COMB. FIELD entry for a field. The entry must be the name of a combined field in the current table.

Correction: Check the entry for misspellings.

>>> Please enter '*' or space

Explanation: Valid entries for the KEY column are either an asterisk (*), which marks the current field as the primary key for the table, or a space for all other fields. You might be in the wrong column.

>>> Please enter a database field name, CTRL-U to backup

Explanation: If you want to build a relationship between the screen field and a database field, you must enter the name of a valid field in this database.

If you want this screen field to accept values only, i.e., as in an SQL screen, you can skip this database field prompt. But you must enter valid type and length values.

>>> Please enter a length, CTRL-U to backup

Explanation: After entering a field type, you must enter a valid length.

Correction: Press CTRL U to return to the field edit prompt. Then enter a database field name.

>>> Please enter a name, CTRL-U to backup

Explanation: The SCREEN: prompt expects a valid screen name.

Correction: If you want to return to the menu, press CTRL-U.

>>> Please enter a screen field name, CTRL-U to backup

Explanation: To enter a screen field you must first enter a name for that screen field.

Correction: Press RETURN. Then enter a screen field name.

>>> Please enter a type, CTRL-U to backup

Explanation: If you do not enter a database field name, you must enter a field type for the screen field being added or modified.

Correction: Press RETURN. Then enter a valid field type. Or, return to the previous prompt, and enter a valid database field name.

>>> Please mount backup media volume nnn. Enter 'Y' to continue, 'N' to abort.

Explanation: This prompt tells you to mount backup tape or diskette volume number nnn.

Correction: To continue, mount the: specified diskette or tape volume and press the volume will begin to read.

>>> Prog/menu unknown

Explanation: The program or menu that you are trying to enter as a menu line cannot be found in the data dictionary.

Correction: Check for any spelling mistakes. You can use Print Menus to get a list of menus, and Print List of Programs to get a list of programs. For more information, see Sections 11.2 and 11.7.

>>> Read error - BUDB

Explanation: There has been an error reading the database or data dictionary. Any keyboard response returns you to the menu.

This usually happens only if someone else is updating the database at the same time you are writing the backup.

Correction: Make sure no one is using the database, and restart the backup.

NOTE: Another reason for this error can be a bad disk block in the database, and the operating system can no longer read the disk. In this case, you must reformat the disk to remove the bad block, then recover the database from a previous backup.

>>> Record not deleted due to related records

Explanation: Other records in the database also reference this record. Therefore, this record cannot be deleted.

Correction: You must delete the related records before you can delete this record.

>>> Record nnn, attempt to modify the key of a record with children.

Explanation: Record nnn of the input file specifies an update to the key field of a database record that has explicit relationships with other records. The record has not been updated.

>>> Record nnnn is a duplicate key.

Explanation: The record number nnn of the input file has a key value that exists in the database. This is an error if you are inserting new records; but it is not an error if you are updating existing records.

>>> Record too large

Explanation: The record being added is too large to fit in ENTER's 2K-byte buffer.

Correction: Reduce the table's record size before trying to maintain it using an ENTER screen.

>>> Record is in use by another process

Explanation: Another user has selected this record. A record can only be selected by one process at a time.

Correction: You must wait. Try to select this record later.

>>> Remove the current transaction log

Explanation: The database has been restored and the current transaction log replayed, so remove the diskette or tape on which the current transaction log is stored.

>>> Requested video mode not supported on this terminal

Explanation: PAINT cannot locate the terminal entries for the requested video mode in your termcap file.

Correction: Modify the termcap file to include the necessary entries.

However, some terminals lack certain video capabilities. Required entries are so and se (reverse video), us and ue (underline), and ru and re (reverse underline). See Section 6.1 for complete termcap information.

>>> Restore complete.

Explanation: The backup has been successfully recovered. Any keyboard response exits the program and returns you to the menu.

>>> Right end of normal area is not on right edge of a video field

Explanation: You are trying to change a video field back to normal mode, but the right end of the normal area does not match the right edge of the existing video field.

Correction: When you change video modes, you must change the entire video field.

>>> Screen already exists

Explanation: You are trying to add a screen form with a name that exists in this database. Screens must have unique names.

Correction: Try entering a new name.

>>> Screen fields cannot be added at a non-blank position

Explanation: There is text in the position where you are trying to add a screen field.

Correction: Add the field at a blank position.

>>> Screen is unknown

Explanation: PAINT cannot locate a screen form in the data dictionary with this name.

Correction: Check the spelling of the screen form name.

>>> Some fields were not put on the screen because there were too many to all fit.

Explanation: The combination of prompts and fields has used up the space on the current table's screen form. If you enter a y at the Continue? prompt, the default screen form is generated using the fields that fit. An n terminates the screen form generation for the current table.

>>> SOL: Can't open xxxx

Explanation: SQL was started from the shell command line to process the script named xxxx. You cannot open this ASCII file.

Correction: Make sure you have access privileges to read the file.

>>> statement on line nnn.

Explanation: This message helps you locate the error that generated the previous RPT error message.

>>> Syntax error

Explanation: The current SQL query contains a syntax error. A message detailing the error follows this message.

>>> syntax error

Explanation: The structure of an RPT command is invalid. This error message is followed by more debugging information.

>>> Table exists

Explanation: You are trying to add a table with the same name as an existing table. All table names must be unique.

Correction: Try entering another name.

>>> Table has no fields

Explanation: The dbcreate program cannot create a screen for tables that don't have fields.

>>> Table is referenced

Explanation: You are trying to delete a table from a database that has one or more of its fields referenced by other tables' fields.

Correction: Remove all references to this table before trying to delete this table.

>>> Table limit exceeded. This program will exit

Explanation: This is an internal system error. There can only be 256 tables per database. You have exceeded this limit.

Correction: To add more tables, you must delete some existing ones.

>>> Table limit will be exceeded

Explanation: There can be only 256 tables per database. If you add another table, you will exceed this limit.

Correction: To add more tables, you must delete some existing ones.

>>> Table: xxxx does not have a key.
The table must have a key if neither -n or -u are given.

Explanation: The default mode for DBLOAD doesn't use flags. This, however, requires the table to have a key.

Correction: If the table does not have a key, use the -n option.

>>> Tables can't have field names

Explanation: You are trying to enter a table name that is already used as a field name in this database. A table name must be unique in the database.

Correction: Enter another name for this table.

>>> Target record key is missing or incomplete

Explanation: All the elements of the target table's key must display on the screen form.

Correction: Make sure you have specified the right target table. If you have, you may have to modify the screen form design.

>>> Termcap entry too long

Explanation: There is not enough memory to load your terminal's termcap description.

Correction: Split the terminal description in your termcap file into two or more physical teminal entries using the following command:

```
tc=terminal_type
```

For more information, see your Operating System Manual termcap description.

>>> Terminal screen is too big, not enough memory.

Explanation: There is not enough memory to display the screen form.

Correction: Change the number of lines and columns specified for your terminal in your termcap file.

>>> TEXT and BINARY fields cannot be keys

Explanation: You are trying to specify a TEXT or BINARY data type as the primary key in the table you are defining.

Correction: Either enter a valid primary key data type, or remove the primary key indicator from this field definition.

>>> TEXT type fields are not allowed.

Explanation: Only NUMERIC, STRING, DATE, LDATE, TIME, FLOAT, or AMOUNT fields can be used in an index definition.

>>> The advanced field attributes file is not present or is not readable.

Explanation: The program cannot find the file, fields.afa, in the current directory.

Correction: Make sure the Advanced Field Attributes file has been developed and that the source file name is spelled correctly. Also check that you have write privileges set for the file, fields.afa.

>>> The backup device environment variable (BUDEV) is not set

Explanation: The environment variable must be set to the name of the backup device. The environment variable is usually set during the installation procedure. Any keyboard response returns you to the menu.

>>> The correct usage is: SQL [-b] [filename]

Explanation: SQL was started from the shell command line with an incorrect number of parameters.

Correction: Enter the command again with the correct parameters.,

>>> The create on intermediate file xxxx failed

Explanation: The intermediate file named xxxx cannot be created. This can be caused by the directory not having enough space, or not having correct group write permission for the user.

>>> The create on tag file xxxx failed

Explanation: The RPT tag file named xxxx cannot be created. This can be because the directory does not have enough space, or the user does not have the correct group write permission.

>>> The database field is not in the same table.

Explanation: The field name you have entered is a valid database field name, but it is not in the same table as the other fields in the index.

Correction: Check the FIELD NAME entry. You can list all tables and their valid field names using Print Database Design.

>>> The database field name is not valid.

Explanation: The field name you have entered is not in the data dictionary.

Correction: Check the FIELD NAME entry. You can list all tables and their valid field names using Print Database Design.

>>> The database has been reconfigured. To restart transaction logging, backup the database

Explanation: The database has been recently reconfigured. This caused transaction logging to be turned OFF.

Correction: To turn transaction logging ON, back up the database.

>>> The field ’ffff' is not in table 'xxxx'.

Explanation: The field you specified is a valid database field, but it is not in the table xxxx.

Invalid Query:
```
select emp.location
from emp/
```

Correction:
```
select dept.location
from dept/
```

>>> The field name, ffff is not in the data dictionary.

Explanation: The field name ffff specified as the default value is not in the data dictionary.

Correction: Make sure the field name is spelled correctly.

>>> The field would not fit in the new position

Explanation: You are trying to transfer a screen field to a blank area that does not accommodate the field's length.

>>> The field would not fit on the screen

Explanation: You are trying to add a screen field in a position that cannot accommodate its length. This usually occurs when you try to add a screen field at the beginning or middle of a line.

>>> The file 'xxxx' already exists.

Explanation: The into clause in the query directs its output into an existing ASCII or binary file. SQL does not overwrite an existing file.

Correction: Either remove the file, or direct the output to a file with a different name.

>>> The file xxxx does not exist

Explanation: This message displays if volmnt cannot find the specified input ordinary file (for file volumes).

Correction: Make sure the path name, the settings of DBPATH and VOLPATH, and spelling of the file name are correct. Also make sure the file suffix

matches the file you created. File volumes can be created using the cat command.

>>> The file xxxx does not exist in /dev

Explanation: This message displays if volmnt cannot find the specified input character or block device (for device volumes).

Correction: Make sure the path name and spelling of the file name are correct. Device volumes can be created using the mknod command.

>>> The file xxxx is neither a character nor block special file

Explanation: volmnt only allows character or block files to be used as the CHARACTER DEVICE name for device volumes. The file name entered is neither a valid character device file nor a valid block device file.

Correction: Check the spelling of the file name. Make sure the specified file is a character device file for the CHARACTER DEVICE field and a block device file for the BLOCK DEVICE field.

>>> The file xxxx is not a block special file

Explanation: volmnt only allows a block file to be used as the BLOCK DEVICE name for device volumes. The file name entered is neither a valid character device file nor a valid block device file.

Correction: Check the file name spelling and file type. Make sure the specified file is a block device file for the BLOCK DEVICE field.

>>> The 'from' clause is missing or invalid.

Explanation: Queries must contain a from clause. This can be caused by various errors. For example, the from clause is missing or its syntax is incorrect. In the following invalid query, the keyword from is misspelled.

Invalid Query:
```
select name
form emp/
```

Correction:
```
select name
from emp/
```

>>> The if clause on line nnn does not contain an integer expression.

Explanation: RPT cannot recognize the expression in the if command on line nnn. You must compare integers to numeric or amount report fields or variables in if statements.

Invalid Script:
```
input
cnumber (numeric 5),
name [string 30],
address [string 30],
state [string 2].
zip [numeric 5]
.
.
.
if zip = '95648' then
.
.
.
```

Correction:
```
input
cnumber [numeric 5),
name [string 30],
address [string 30],
state [string 2],
zip [numeric 5]
.
.
.
if zip = 95648 then
.
.
.
```

The indexes will not be usable until the preceding error is corrected and this program finishes successfully. If the database is updated before the indexes are usable, they will have to be rebuilt.

Explanation: This message displays in two parts if the program encounters trouble just before it finishes processing. If you can fix the problem before any database updates are done, run this option again.

>>> The input value is not within any of the given ranges.

Explanation: This is the standard error message that displays when the entered value for a field is not a legal value. Legal values are set using the Advanced Field Attributes feature.

Correction: Check the value for mistakes.

>>> The maximum number of indexes (255) has been reached. Another index cannot be added to the system.

Explanation: This message displays if 255 indexes are defined and you are adding another.

Correction: Before you can add another B-tree index, you must drop an existing B-tree index.

>>> The menu cannot be found in the data dictionary

Explanation: This menu has been recently deleted and cannot be found in the data dictionary.

>>> The menu is now full. The remaining screens will be created, but not placed on the menu

Explanation: 16 screens have been created and placed on the Data Entry Menu (usermenu). No room is left to add the remaining screen forms. However, these screens will be created and registered with ENTER.

>>> The new expected number must be at least nnn

Explanation: This message displays if a database file exists, and you try to reduce the EXPECTED number below the maximum number of records existing at one time in this table.

You cannot reduce the EXPECTED number below the value of nnn. Notice that the value of nnn can be greater than the current number of records in the specified table. This is because deleted record slots still take up space in the database file.

Correction: To recover this space, you must perform the following steps:

1. Using SQL, dump all records of the table you want to compress to an ASCII file.
2. Add a new table with the same design as the table you want to compress.
3. Delete the existing table from the database design.
4. Reconfigure the database.
5. Load the data back into the new table.
6. To create a new empty database file with lower EXPECTED numbers, remove file.db and file.dbr. You can then enter the desired EXPECTED number.

>>> The open on intermediate file xxxx failed

Explanation: RPT cannot open the file xxxx that it created previously. Make sure the file still exists; it may have been removed by another user.

>>> The open on tag file xxxx failed

Explanation: RPT cannot open the tag file that it created previously. Make sure the file still exists; it may have been removed by another user.

>>> The referred-to sort expression is xxxx.

Explanation: The sort expression referred to in the previous error message is listed as xxxx. This message helps you locate the previous message's error in the report script.

>>> The size or type of the default is invalid for this field.

Explanation: You are using another field's value as the default in the specified field statement, but the data types or sizes of the two fields do not match.

Correction: Make sure the field name is spelled correctly.

>>> The standard Unify DataServer/ELS menu "mainmenu" does not exist The menu "usermenu" will not be created for the data entry screens

Explanation: The Unify DataServer/ELS Main Menu cannot be found in the data dictionary. It may have been removed.

Correction: Use Add, Modify or Delete Menus to check for the Main Menu and add the menu back if necessary.

>>> The String table has overflowed.

Explanation: The capacity of the string constant table has been exceeded.

Correction: Reduce the number or size of the string constants in the script, or increase the size of the RPTCONCNT environment variable.

>>> The table xxxx, is not in the data dictionary.

Explanation: The specified table xxxx is not in the data dictionary.

Correction: Make sure the table name is spelled correctly.

>>> The table is not valid.

Explanation: The table name you have entered is not in the data dictionary.

Correction: Recheck the TABLE entry for misspellings. You can list all tables using Print Database Design.

>>> The TERM environment variable is not set

Explanation: The TERM variable has not been set. There isn't a default value for this variable.

Correction: Set the TERM variable for your terminal type.

>>> The terminal type, xxxx, is not in the termcap file

Explanation: The terminal type set by your TERM environment variable cannot be found in the termcap file named by TERMCAP.

Correction: Check for correct representation of your terminal's type.

>>> The transaction journal could not be opened

Explanation: The transaction logging file or device cannot be opened.

Correction: Make sure you have read and write privileges for the file.

>>> The transaction log and the database have different version tags

Explanation: This error occurs if the transaction log and the database appear to be different versions. You are trying to replay the wrong transaction log.

>>> The transaction log is now 100% full. - Transaction logging is now off.

Explanation: The transaction logging file has reached is maximum size. No room is left to record changes to the database. Therefore, transaction logging has been turned OFF.

Correction: Perform no more updates, and back up the database. Transaction logging can only be restarted by running Write Database Backup.

>>> The transaction log is now nnn% full.

Explanation: This message displays as the transaction logging file fills up, updating for 50, 80, and 90 percent. Transaction logging is turned off at 100%.

>>> The value of variable: xxxx is never set.

Explanation: The value of xxxx has not been initialized with the set command. Or, xxxx has not been included as a report field in the input section.

Otherwise, xxxx can be misspelled or it can be a heading without the required quotes.

Invalid Script:
```
input
name (string 30],
number [numeric 5)
before report
print Customers
detail
print name, number
end
```

Correction:
```
input
name [string 30],
number [numeric 5]
before report
print 'Customers'
detail
print name, number
end
```

>>> The variable type is vvvv and the expression type is xxxx.

Explanation: You cannot assign a new value to a variable that has been set if the values' data types are not compatible.

Invalid Script:
```
set z_variable to 4.89
.
.
.
set z_variable to 'abc' + 'def'
```

Correction:
```
set z_variable to 4.89
.
.
.
set new_variable to 'abc' + 'def'
```

>>> The volume type ’D' needs a block special file as its block device

Explanation: This message displays after a device volume definition is completed, when the volume's block device is not a block special file in the /dev directory.

Correction: Change the block device name to an existing block device in the /dev directory.

>>> The volume type 'D' needs a special file as its character device

Explanation: This message displays after a device volume definition is completed, when the volume's character device is neither a block nor a character special file in the /dev directory.

Correction: Change the character device name to an existing block or character device in the /dev directory.

>>> The volume type ’F’ must have matching block and character devices

Explanation: This message displays after a file volume definition is completed, if the files specified for block and character devices do not match.

Correction: Change the block device name and make sure the same name appears as the character device name.

>>> The volume type ’F’ needs a regular file as its block device

Explanation: This message displays after a file volume definition is completed, when the volume definition is type F, for a file volume, but the block device name is not a regular (ordinary) file.

Correction: Change the block device name to a valid regular file name.

>>> The volume type ’F’ needs file.db or $DBNAME as its block device

Explanation: This message displays after a volume definition is completed for a root volume of type F (file volume), if the block device name is neither file.db nor the name specified in the environment variable DBNAME.

Correction: Redefine the root file volume. The program should fill in the block device name as file.db,or as the name specified in DBNAME, if the environment variable is set.

>>> The write to FNDFLD failed errno - nnn

Explanation: RPT cannot write to the FNDFLD program. This error message usually accompanies other error messages.

Correction: Correct all other errors found. Then, make sure the UNIFY environment variable is set correctly and that there is correct write permission on the directory. If this is the only message and the problem persists, contact Unify Corporation.

>>> The xxxx table has overflowed.

Explanation: The table xxxx has exceeded its maximum number of elements.

Correction: Reduce the number of elements of type xxxx in the script, or increase the size of the table using the appropriate environment variable. You may need to reduce the size of another table to increase the size of this one.

The different tables and their maximum values are listed in the RPT Table Usage Information Report.

>>> There are no B-tree indexes to rebuild.

Explanation: The data dictionary has no B-tree index definitions.

>>> There are no fields defined in the database design

Explanation: This message displays if a database design exists but contains tables with no fields. For tables to be compiled, they must have fields.

Correction: Check your environment variables or current directory to make sure that you are referencing the correct database. Use either Design and Create a New Database or Modify Database Design to correct your table definitions.

>>> There are no indexes for the given table.

Explanation: The table name you have entered has no indexes defined.

Correction: Make sure the table name is spelled correctly. Use Print or Display B-Tree Statistics to list the current B-Tree index definitions.

>>> There are no input records

Explanation: The input file has no data. If you are using SQL to select the data, try running the query by itself to make sure it produces results.

>>> There are no more indexes in the set.

Explanation: You have just deleted the last index for this table.

>>> There are no screens defined

Explanation: You are entering a screen name or ALL at the SCREEN: prompt, but there are no screen forms defined for this database.

>>> There are too many saved strings.

Explanation: This internal error occurs when too many strings are formatted with the using option of print.

Correction: Reduce the number of using formats in the report print statements.

>>> There is a command group which refers to a non-existent sort expression on line nnn.

Explanation: The report script is trying to refer to a name in a before or after command that is not listed as a sort element.

Invalid Script:

```
.
.
.
sort customer_num
before zip_code
print 'Zip Code Area
```

```
.
.
.
```

Correction:

```
.
.
.
sort customer_num, zip_code
before zip_code
print 'Zip Code Area'
.
.
.
```

>>> There is a nested aggregate operator on line nnn.

Explanation: Aggregate functions cannot be used in a sort expression.

Invalid Script:

```
print total(new_amount + total(old_amount))
```

Correction:

```
set total_old to total(old_amount)
print total(new_amount + total_old)
```

>>> There is a phase error in the input file.

Explanation: The format of an input line does not match the input section of the script.

Correction: Check for extra values on each line of the input. Make sure that there is a value for each input column in the input section. Also verify that the type of the value and the type of the input column match. Look for unescaped separator characters embedded in a column value.

>>> There is a saved string overflow.

Explanation: This internal error occurs when an excessively long string is formatted by the using option of print.

Correction: Try reducing the length of the input string.

>>> There is a type mismatch in the set clause on line nnn

Explanation: You cannot assign a new value to a variable that has been set if the values' data types are not compatible.

Invalid Script:
```
.
.
.
Set z to 1.34
if employ_code = 3 then
set z to 'abc' + 'efg'
```

Correction:
```
.
.
.
set z to 1.34
if employ_code = 3 then
set string_variable to 'abc' + 'efg'
```

>>> There is a variable used in a sort expression on line nnn.

Explanation: You cannot use variables as sort elements. Make sure you have not used variables as sort elements. Otherwise, make sure you have no misspellings in the command line.

Invalid Script:
```
sort czip, x
.
.
.
before report
set x to 0
```

>>> There is an aggregate expression embedded in a sort expression on line nnn.

Explanation: Aggregate functions cannot be used in a sort expression.

Invalid Script:
```
sort czip, total(x_amount)
```

>>> There is an assignment to a non-variable on line nnn.

Explanation: You cannot use an alphabetic string as a variable and as a report field name in the same report script.

Invalid Script:
```
input
x [date),
name [string 30]
before report
set x to 4
.
.
.
```

Correction:
```
input
o_date [date],
name [string 30]
before report
set x to 4
.
.
.
```

>>> There is an error in an 'if' expression.

Explanation: RPT has found an incorrect expression in the if command.

Correction: Review correct RPT expression syntax in Section 17.4.

>>> There is an integer constant expected for the length.

Explanation: When a data type and length is assigned to a report field in the input section, LENGTH must be an integer. See Sections 17.2 and 17.7.

Invalid Script:
```
input
customer_num [numeric xx],
.
.
.
```

Correction:

```
input
customer_num [numeric 12],
.
.
.
```

>>> There is an invalid aggregate expression

Explanation: The syntax for an aggregate function is incorrect. See Section 17.5.

Invalid Script:

```
print total(total)
print total(cust_amount
where cust_amount > 7.00)
```

Correction:

```
print total(correct_field)
print total(cust_amount
where cust_amount > 7.00)
```

>>> There is an invalid command

Explanation: RPT do0es not recognize the word located on line nnn as a command-group identifier. The next error message indicates the error line number.

Often this error is caused by a misspelling of the first word on a non-command-group line following a command-group command.

Invalid Script:

```
.
.
.
before report
ssset X to 0
```

Correction:

```
.
.
.
before report
set x to 0
```

>>> There is an invalid column clause

Explanation: An integer constant or variable must follow the in column clause. This error message often occurs when a single special character is detected in the column value field or a column value has been omitted.

Invalid Script:
```
print zip_code in column
```

Correction:
```
set zip_column to 20
.
.
.
print zip_code in column zip_column
```

>>> There Is an invalid function argument list

Explanation: The arguments for a function have incorrect syntax. Such syntax errors include a missing parenthesis, a missing comma, or no matching single quote.

Invalid Script:
```
input
employ_num [numeric 8],
employ_code [numeric 1],
.
.
.
set level_num to <1_num>
index(employ_code, ’levl', 'lev2' 'lev3')
```

Correction:
```
input
employ_num [numeric 8],
employ_code [numeric 1],
.
.
.
set level_num to <1_num>
index(employ_code, ’lev1', 'lev2', 'lev3')
```

>>> There is an invalid input item

Explanation: RPT does not recognize a report field name in the input command section. RPT keywords or function names cannot be used as input report field names. This error can also be caused by including a comma after the last input item. RPT then reads the next line as part of the input section.

Invalid Script:
```
input
order_num [numeric 9],
.
.
.
total [amount 8]
```

Correction:
```
input
order_num [numeric 9],
.
.
.
order_total (amount 8]
```

>>> There is an invalid non-command-group command

Explanation: Make sure the correct syntax is used for the non-command group. The causes of this error include the following:

- misspelling a command word
- omitting a comma
- trying to print a multi-word heading without single quotes
- omitting the end command to terminate the report script
- not putting the set command within a command-group

Invalid Script:
```
input
name [string 30),
number [numeric 5]
beffore report
.
```

```
.
.
```

Correction:

```
input
```

Correction:

```
input
name [string 30],
number [numeric 5]
before report
.
.
.
```

>>> There is an invalid print item

Explanation: The print expression has an incorrect syntax. Note that you cannot use keywords in print expressions.

Invalid Script:

```
print report (report is a keyword)
```

>>> There is an invalid type expression

Explanation: RPT does not recognize the data type in a report field listed under input. Verify the spelling of the word numeric, float, amount, date, ldate, time, string, or text.

COMB is not a valid data type.

Invalid Script:

```
.
.
.
model_num [Numeric 20],
```

Correction:

```
.
.
.
model_num [numeric 20],
```

>>> There is insufficient memory for the record buffer

Explanation: The allocated record buffer doesn't have enough room to read the current input record data.

Correction: Reduce the size of the record.

>>> There is insufficient memory for the input fields.

Explanation: There is not enough memory to process your report.

Correction: Increase the size of the input buffer using the RPTINBUFSZ environment variable. You may need to decrease another table size as well.

Or, reduce the total memory requirements by decreasing the lengths of output strings and report fields, decreasing the number of variables, or reducing the complexity of the report script.

>>> There is insufficient memory for the output line buffer.

Explanation: There is not enough memory left for the buffer to print an output line.

Correction: Use the width command to decrease the output width, if possible. The default width is 132 characters.

Or, reset the RPT environment variables to decrease the amount of memory allocated for the various tables.

>>> There is insufficient memory to hold variables

Explanation: There is not enough internal memory left to store all the variable values.

Correction: Increase the number of variables allowed by using the RPTFLDCNT environment variable. Or, reduce total memory requirements by decreasing the lengths of output strings and report fields, decreasing the number of variables, or reducing the complexity of the report script.

>>> There is insufficient memory to store the constants.

Explanation: There is not enough internal memory left to store all data associated with the print statements.

Correction: Increase the number of constants allowed by using the RPTCONCNT environment variable. Or, reduce the total memory requirements by decreasing the lengths of output strings and report fields, decreasing the number of variables, or by reducing the complexity of the report script.

>>> There is no current field to delete

Explanation: There isn't a screen field in this position to delete.

>>> There is no current field to transfer

Explanation: There isn't a screen field in this position to transfer.

>>> There is no field to modify

Explanation: There isn't a screen field in this position to modify.

>>> There is no help file for this field

Explanation: You are asking to display a help file for the current field, but no help file has been specified for this field.

Correction: You can assign a help file to an individual field using the Advanced Field Attributes option. For more information on Advanced Field Attributes, see Section 8.6.

>>> There is no index with the given ID number.

Explanation: The index ID number you have entered is not in the data dictionary.

Correction: Check the index ID number entry. You can list all index ID numbers and their associated tables using Print or Display B-Tree Statistics.

>>> There is no logical transaction group current

Explanation: Because of an internal error, a "start logical transaction" entry has not been recorded for the present transaction process.

>>> There is no more room in the database for records of this type. Delete some records or increase the expected number and reconfigure.

Explanation: You have reached the maximum number of records you can add to this table, given the existing expected number setting.

Correction: To add more records, you must either delete some records from this table or, using Modify Database Design, increase the EXPECTED value for this table.

>>> There is no more space for this table. There were nnn record(s) added, mmm record(s) updated.

Explanation: The database has no space left to add records to the current table. This occurs when the database has 66% more records than expected by the database design.

Correction: Either delete some records, or use Modify Database Design to increase the expected number of records in the table. Then reconfigure the database.

>>> There is no more space on the standard Unify DataServer/ELS menu "mainmenu". The menu "usermenu" will not be created for the data entry screens.

Explanation: The Unify DataServer/ELS Main Menu already contains 16 selections. No room remains to add the Data Entry Screens menu.

Correction: If you want a Data Entry Screen, use Add, Modify or Delete Menus to reorganize the Main Menu and free a menu line.

>>> There is no previous menu

Explanation: You are at your home menu.

>>> There was an error creating an index file

Explanation: A B-tree index file could not be created.

Correction: Make sure you have read and write privileges for all B-tree index files, for example, bt001.idx. Also make sure you have permission to create files in the database bin directory.

>>> There was an error writing to the disk

Explanation: The data dictionary and database file could not be recovered. The data dictionary and database are now invalid.

Correction: Any keyboard response returns you to the menu. To recover, either read this backup again, or try a different one. Make sure you have enough disk space for the files.

>>> This definition is the same as the one for index number nnn. Do you want to change this definition? (If you don't it will be discarded.)

Explanation: You have entered a duplicate of index number nnn.

Correction: Discard this definition or change it.

>>> This definition specifies that duplicates are not allowed, but the database contains duplicates.

Explanation: Your attempt to add or rebuild a B-tree index failed because the definition specifies that duplicate entries must not exist in the index. However, the database contains field values that can cause duplicate index entries.

Correction: Make sure all database fields listed in the B-tree index definition have unique values in each record. Also make sure no records have indexed fields that are blank. Default values or blank fields in more than one record would be duplicates.

>>> This field is too big for the B-tree key as defined. Its internal length is nnn, and only mmm more will fit.

Explanation: You are trying to add another field with length nnn to the B-tree index key, but there are only mmm bytes left. The maximum length is shown at the end of the prompt, THE SUM OF INTERNAL LENGTHS...

>>> This field is write protected. Its value cannot be changed.

Explanation: You cannot change this field because it is write-protected.

Correction: You must provide the write password for this field before you can change its value.

>>> This is the current menu

Explanation: You are trying to select your current menu.

>>> This is your home menu

Explanation: The home command key cannot work because you are at your home menu.

>>> This record already exists

Explanation: You are trying to enter a primary key value that exists in the database. Primary key field values must be unique.

Correction: Use another value for this field entry.

>>> This record has been deleted

Explanation: Since the time you selected this record, the record has been deleted by another process.

Correction: Even though the record is listed in your selection file, it is no longer in the database.

>>> This value is a duplicate. Please enter a different value

Explanation: Either the field is a key field, or the field is indexed in a B-tree that does not allow duplicate entries (it is a secondary key). The same field in another record in this table has this value.

Correction: Either change the other field, or enter a different value for this field.

>>> This was detected in a set expression on line nnn.

Explanation: This message helps you locate the error that generated the previous error message.

>>> This was detected in the column expression for the print statement on line nnn.

Explanation: This message helps you locate the error that generated the previous error message.

>>> This was detected in the expression on line nnn.

Explanation: This message helps you locate the error that generated the previous error message.

>>> This was detected in the if expression on line nnn

Explanation: This message helps you locate the error that generated the previous error message.

>>> This was detected on line nnn

Explanation: This message helps locate the error that generated the previous error message.

>>> To move the current field, position the cursor and hit the TRANSFER FIELD key

Explanation: The standard PAINT command key for this operation is T. Your unicap file can be set differently.

To transfer the field highlighted by the mark character, make sure there is enough space, move the cursor to the new position, and press the transfer field command key again.

NOTE: You cannot move lines that contain wraparound screen fields.

>>> To set normal video, position the cursor and hit the NORMAL VIDEO key

Explanation: The standard PAINT command key for this operation is n. Your unicap file can be set differently.

Correction: Move the cursor to the end of the area you want to display in normal video and press the normal video command key again.

>>> To set reverse video, position the cursor and hit the REVERSE VIDEO key

Explanation: The standard PAINT command key for this operation is [. Your, unicap file can be set differently.

Correction: Move the cursor to the end of the area you want to display in reverse video and press the reverse video command key again.

>>> To set rv + ul video, position the cursor and hit the RV + UL VIDEO key

Explanation: The standard PAINT command key for this operation is +. Your unicap file can be set differently.

Correction: Move the cursor to the end of the area you want to display in reverse underline video mode and press the reverse underline video command key again.

>>> To set ul video, position the cursor and hit the UNDERLINE VIDEO key

Explanation: The standard PAINT command key for this operation is]. Your unicap file can be set differently.

Correction: Move the cursor to the end of the area you want to underline and press the underline command key again.

>>> Too few blocks per media, increase value of BUTAPESZ environment variable

Explanation: So few blocks are specified in the BUTAPESZ environment variable that no data can be read from or written to the backup tape or diskette.

Correction: Reset the BUTAPESZ environment variable to the number of blocks on the backup tape or diskette and try again.

>>> Too few blocks per media, increase value of BUTAPESZ environment variable.

Explanation: The number of blocks specified in the BUTAPESZ environment variable is less than the number of blocks on the tape or diskette volume.

Correction: Reset the BUTAPESZ environment variable to match the number of blocks on the backup media and try again.

>>> Transaction logging has just been turned on

Explanation: Transaction logging is now active because a new backup has been made.

The transaction log file has been reset and transaction logging has been turned on.

>>> Transaction logging is currently off

Explanation: Transaction logging has not been turned on because either a new backup has not been made, or the requested logging mode is set to OFF.

>>> Transaction logging is now off. Backup the data bass to restart it.

Explanation: If you are using the transaction logging feature of Unify DataServer/ELS, reconfiguration of the database causes logging to be turned off.

Correction: You must back up your database to restart transaction logging.

>>> Transaction logging will be turned on once the database is backed up.

Explanation: After you set transaction logging, if you have updated the database, you must back up the database before transaction logging can be turned on.

>>> Transaction logging is off due to an error. Backup the database to restart transaction logging.

Explanation: Transactions cannot be written to the log file. This can be caused by a write error, a file permission problem, or a log file that's full. You must back up the database before transaction logging can be turned ON.

>>> Transactions are currently being logged.

Explanation: The transaction logging status is ON.

>>> Trying to initialize command prompt lines

Explanation: While trying to set up the Unify DataServer/ELS command prompt lines, the operating system ran out of memory. This often signals a hardware problem. You can get second level help messages by responding to the error message with?.

>>> Trying to initialize virtual characters

Explanation: While trying to set up the Unify DataServer/ELS virtual characters, the operating system ran out of memory. This often signals a hardware problem. You can get second level help messagts by responding to the error message with?.

>>> Type and length are incomplete. Delete?

Explanation: A valid field name is present but no value is set for the field’s data type or length. These two parameters must be set for a field to be valid.

Correction: Either enter the values for this field description, or answer y (yes) and delete the current field.

>>> Type conversion error.
Left type is LLLL, right type is RRRR, operator is xxxx.

Explanation: The data types of the two sides of a Boolean expression do not match. They cannot be converted into compatible data types. You may have used the wrong field name.

Invalid Query:
```
select name
from emp
where name = 2000/
```

Correction:
```
select name
from emp
where salary = 2000/
```

Explanation: The data types of the operands cannot be converted so the data types match. This error message is followed by more debugging information.

>>> Unable to create output file, please reenter filename

Explanation: During the Report Option process, the given file name (FILENAME) could not be created.

Correction: Make sure you have spelled the file name correctly. Or Make sure you have write permission for the file and directory.

>>> Unable to create the unify.conf configuration file.

Explanation: The transaction logging program cannot create the file named unify.conf. This file stores the Unify DataServer/ELS transaction logging configuration. The program exits to the active menu.

Correction: Check the write privileges for files in the database directory.

>>> Unable to execute FNDFLD - nnn.

Explanation: Make sure the PATH and Unify DataServer/ELS environment variables are set correctly, and exported if necessary (Section 5).

>>> Unable to execute the database back up program.

Explanation: This is an internal system error.

Correction: Make sure the database backup program has not been deleted or moved to a different directory, or had its file permission modes changed.

>>> Unable to execute the replay program.

Explanation: This is an internal system error.

Correction: Make sure the database backup program has not been deleted or moved to a different directory, or had its file permission modes changed.

>>> Unable to execute the sort command

Explanation: Make sure the PATH and Unify DataServer/ELS environment variables are set correctly and exported if necessary (Section 5).

>>> Unable to find help documentation for this topic

Explanation: There is no help text file available for the current menu.

>>> Unable to open xxxx

Explanation: SQL was started from the shell command line to run the script named xxxx. You cannot access this file.

Correction: Make sure you have correct access privileges to read the file.

>>> Unable to open xxxx

Explanation: This message is sent to the errlog file when the database is started if you have defined a root volume as a file volume but you have not specified the VOLPATH environment variable. For more information about file volumes, see Define Database Volumes, Section 9.5.

Correction: Make sure you have set the VOLPATH environment variable.

>>> Unable to open backup device

Explanation: Read Database Backup or Write Database Backup cannot open the specified backup device.

Correction: Make sure the BUDEV environment variable is set correctly. Also, make sure you have correct access privileges for the backup device.

>>> Unable to open database file

Explanation: You have used an incorrect database file name.

Correction: Check spellings of the name and make sure you have read privileges. Also make sure that you are in the proper directory. The file name must be file.db or unify.db.

>>> Unable to open database volume

Explanation: One of the database volumes specified in Define Database Volumes cannot be opened. The database and data dictionary are now invalid.

Correction: Two reasons for this error are that you don't have write privileges for the volume, or that the special file entry has been removed or corrupted.

Any keyboard response returns you to the menu. For Read Database Backup, to recover, you must read the backup again after you have corrected the problem with the volume.

>>> Unable to open file: xxxx.

Explanation: The ASCII file named xxxx cannot be opened or located.

Correction: Make sure you have read permission for the file. Also, make sure the file exists and the file name is spelled correctly.

>>> Unable to open index file

Explanation: One of the B-tree index files cannot be opened.

Correction: Make sure your operating system user ID has read/write privileges for all B-tree index files. These are the files with such names as bt001.idx.

>>> Unable to open index information file

Explanation: The file, field.idx, cannot be opened.

Correction: Make sure you have read/write privileges for this file.

>>> Unable to open pipe.

Explanation: RPT has detected an internal error.

Correction: Try running the report again. If the problem persists, contact Unify Corporation.

>>> Unable to open script file xxxx.

Explanation: RPT cannot find or open the report script file xxxx. Check for a misspelling of the report script file name or for incorrect file permission.

Invalid Script: `RPT rptsrpttt`

Correction: `RPT rptscript`

>>> Unable to open specified input file xxxx.

Explanation: RPT cannot find or open the input file named xxxx. Check for a misspelling of the file name or for incorrect read permission.

Invalid Script: `RPT rptscript inppputfile`

Correction: `RPT rptscript inputfile`

>>> Unable to open the configuration file

Explanation: This error occurs if the configuration file cannot be found or read. The replay program needs the information in this file to work.

Correction: Check your file permission modes on unify.conf in the database bin directory.

>>> Unable to open the incomplete ltg file

Explanation: The incomplete transaction group summary file cannot be created.

Correction: Check your permission modes on the file or directory.

>>> Unable to open the replay error file

Explanation: The replay error file cannot be created.

Correction: Check your permission modes on the file or directory.

>>> Unable to open the termcap file

Explanation: The termcap file could not be opened.

Correction: If you have the TERMCAP environment variable set, make sure that the specified file exists and is readable. If you don't have TERMCAP set, make sure /etc/termcap exists and is readable.

>>> Unable to open the transaction logging device

Explanation: Transaction logging is set to ON and ENTER cannot open the transaction logging device specified in Transaction Logging Status to record transactions.

Correction: Make sure the device is online or the log file exists. Also make sure you have proper permissions/access modes to write to the device or file.

>>> Unable to process screen. Screen .q and .h files not writeable

Explanation: You are trying to save a screen form which exists, but you don't have permission to overwrite the existing q and h files.

Correction: Correct the file permissions before trying again.

>>> Unable to rebuild the index. It must be dropped or successfully rebuilt before the database table involved can be accessed.

Explanation: The most common reason the index cannot be rebuilt is the file system has run out of space. Otherwise, you may have incorrect permissions for the database directory specified in DBPATH or the idxmnt program cannot run the BLDBI executable.

Correction: Either drop the index or fix the problem and try again. If you want to try again, write down the definition and drop the index and then fix the problem. You can quickly add the index back later.

To determine the problem, check to see how much file system space is left. If this is the problem, you can gain more space by deleting unwanted files.

Otherwise, make sure you have read and write privileges for the files in the database directory. Or make sure the executable BLDBI is located in the unify/bin directory.

>>> Unable to set an exit command key due to a conflict with another command.

Explanation: You must have an exit command key defined or you can't exit from Unify DataServer/ELS. No exit command key is defined in the unicap file, AND the default value for the exit key (^D) is used for some other Menu Handler command.

Correction: Define an exit command key OR change the command definition that currently uses ^D.

>>> Unable -to start the compiler

Explanation: The compiler cannot begin.

Correction: Make sure the compiler afac is on your PATH.

>>> Unable to start up the editor

Explanation: The editor cannot begin.

Correction: Check that the editor is on your PATH, or make sure you have set the EDIT environment variable correctly.

>>> Unable to write the transaction log

Explanation: ENTER cannot write to the transaction log because of a permission/ access mode problem.

Correction: Change the permission so you can write to the file.

>>> Undefined return from pfield

Explanation: This is an internal system error.

Correction: Call Unify Corporation.

>>> Unexpected command type xxxx - docom

Explanation: RPT has detected an internal error.

Correction: Try running the report again. If the problem persists, contact Unify Corporation.

>>> Unexpected string operator xxxx - oexwdth.

Explanation: RPT has detected an internal error.

Correction: Try running the report again. If the problem persists, contact Unify Corporation.

>>> Unexpected type comptyp XXXX

Explanation: RPT has detected an internal error.

Correction: Try running the report again. If the problem persists, contact Unify Corporation.

>>> Unexpected type in intsize xxxx

Explanation: RPT has detected an internal error.

Correction: Try running the report again. If the problem persists, contact Unify Corporation.

>>> Unify DataServer/ELS reserved words are not allowed as names

Explanation: Modify Database Design does not allow Unify DataServer/ELS reserved words to be used as field or table names. These reserved words are mostly RPT and SQL keywords.

Correction: Check the spelling of your entry or use another name. For a list of Unify DataServer/ELS reserved words, see Appendix B.

>>> Unmount the transaction log

Explanation: Because you have set the log file to a device, the current transaction log volume must be unmounted before the backup process can continue. This assumes that you have only one backup device.

>>> Unrecognized character: 'x'

Explanation: RPT has detected a character that it does not recognize. This message can be the result of a missing single quote or an inserted invisible control character.

>>> Unrecognized sql command

Explanation: SQL cannot recognize the first keyword of the command.

Correction: Make sure you have spelled the keyword correctly.

>>> usage: DBLOAD [-n|-u|] [-l] [-s separator] [-b] <database> <table> <file> <spec-file>

Explanation: You haven't supplied the correct number of parameters to DBLOAD.

Correction: Use the syntax in the error message to try again.

NOTE: You cannot use the parameters -n and -u together. Values must be entered for the existing records before adding this one.

>>> Values must be entered for the existing records before adding this one.

Explanation: This record cannot be added because this table has fields belonging to a "no duplicates" B-tree index. One or more of the fields in the previously added record still have default values.

All- fields in a no duplicates B-tree index must have unique values or future records cannot be added to that table.

Correction: Go back and enter unique values for those fields. After those fields have values, you can add new records.

>>> Warning: 'group by' and 'order by' in the same query. The 'order by' will be ignored.

Explanation: The group by and order by clauses cannot be used in the same query. This error message is only a warning that the order by will not be performed. SQL ignores the order by and runs the query.

>>> Warning: Unable to verify volume length.

Explanation: This message displays if the length of the raw device cannot be verified. This is a warning message only.

Correction: Verify that the disk partition contains the number of blocks specified.

>>> Warning: 'unique' may not be used with aggregate functions. The 'unique' specification will be ignored.

Explanation: The unique specification cannot be used with the aggregate functions min, max, sum, avg, or count. SQL runs the query, but does not eliminate duplicate rows.

>>> Write error - BUDB

Explanation: An error has occurred while writing the backup diskette or tape. Any keyboard response returns you to the menu.

Correction: Make sure the number of blocks per volume is set correctly. The number may be too large, depending on the hardware. Use Modify System Parameters & Security to reduce the number of blocks.

If this error has occurred, the tape or diskette has probably been damaged. Start the backup over again, using a new tape or diskette.

>>> Write error - REDB

Explanation: An error has occurred while writing the backup to the disk. The database and data dictionary are now invalid. You probably have run out of space on the disk.

Correction: Make sure you have enough space on the disk. Then try reading the backup again. Any keyboard response returns you to the menu.

>>> You are not allowed to execute any options on that menu

Explanation: Even though you have access privileges to the selected menu, you do not have access privileges to the options listed on that menu.

Appendix E: The licprod utility

This appendix explains how to perform licensing in two conditions.

- after installation
- during installation

The licensing program activates these ACCELL/SQL files depending on the options and additional products you purchased.

Executable /Archive	Description
ENTER	UNIFY ENTER screen manager
SCHENT	Unify Datraserver/ELS database schema development utility
SQL	Unify Datraserver/ELS SQL manager
cmn.a	Common release archive

Licensing after installation

To prepare for licensing your software after installation, complete these steps:

Step 1 Make sure that the Unify DataServer/ELS configuration variable is set to the full pathname of the release lib directory.

Step 2 Make sure that the PATH configuration variable contains the full pathname of the release bin directory.

Step 3 Start licprod.

❖ licprod [-version I -f filename) [unify]

-version Directs the utility to display the release version of licprod that you are using.

-f filename — Directs the utility to retrieve the license key from the file named filename. If you do not use the -f option, licprod will use install/license.key if the file contains data. Otherwise, licprod prompts you for the license key.

unify — License only the Unify DataServer/ELS DBMS executables

Step 4 — If you do not use the -f option, the utility prompts you for a license key with this message:

```
Enter your license key (’sh’ for shell, 'x' to exit):
```

❖ Enter your license key, or answer appropriately as described below:

X: — If you do not have a license key code, enter "x". You see this message and the utility ends:

```
User requested exit.
```

sh: — If you want to escape to the operating system, enter "sh". You see this message:

```
Pushing shell... Type 'exit' to return.
```

To return to this step in the licensing process, enter "exit" at the operating system prompt.

Step 5 — If you do not specify a product to license (unify), you are prompted for a product choice:

```
Select one of the following products to license.

1) UNIFY database.
2) ACCEWIDS Runtime
3) ACCELL/IDS Development.
selection:
```

❖ 1

The licensing procedure continues.

Step 6 — Check the status of your release's licensing by using the prlcinf utility.

- unsup/pricinf [bin/executable_name]

  See the table at the beginning of this appendix for a list of executable names

Licensing during installation

To license your software release during the product installation process, complete these steps:

Step 1 If you are licensing a release for the first time, select an appropriate licensing option from this menu:

```
A licensing key has not been entered in the file ’install
/license.key’

Select one of the following licensing options:

1. Enter license key now.
2. Continue installation and license the release later.
3. Terminate installation.
Selection:
```

Enter license key

If you select " 1 ", you can enter your license key code when the utility prompts you with this message:

```
Enter your license key ('sh’ for shell, ’x’  to exit):
```

- Answer appropriately as described below:

  key.- Enter your license key.

  X: If you do not have a license key code or want to skip the software licensing, enter 'Y'. You see the following message. To continue the installation, return to step 18.

```
Not installing license. Installation continuing..
```

  sh: If you want to escape to the operating system, enter "sh". You see this message:

```
Pushing shell... Type 'exit' to return.
```

To return to this spot in the installation process, enter "exit" at the operating system prompt.

- When you enter a license key, you are asked to verify the key you entered with this message:

```
License key entered: xxxxxxxxxxxxxxxx
Use? (y/[n])
```

- Respond appropriately to the prompt as described below:

No: If you answer with "n", you will be given another opportunity to enter the license key.

Yes: If you answer "yes," the key value is placed in the install/license.key file.

The licensing program then licenses the appropriate executables depending on the options and additional products you purchased. You see the message:

```
Preparing to license your product.
```

As the license key is being installed in the appropriate executables, the following message is added to the Installmsgs file for each executable that is licensed:

```
License code 'xxxx' has been installed for executable
xxxx.
```

When the licensing is complete, you see this message:

```
Licensing procedure successful.
```

Continue installation.

If you select 2, you see these messages:

```
Not installing license.
A license may be obtained by contacting unify
Client Services.
Installation continuing ...
```

You will not be able to use Unify DataServer/ELS DBMS until you license it. You can license the software after the release installation is finished by using the licprod utility described in "licensing After Installation".

Terminate installation

If you select 3, you see this message:

```
Installation interrupted by user.
```

The installation program ends. You cannot use ACCELL Unify DataServer/ELS DBMS until you rerun Install/install and license the software.

Step 2 If you are installing the Unify DataServer/ELS DBMS software with the install/install utility, continue the installation by returning to step 18.

Appendix F: Tuning your operating system

You should review your operating system configuration to determine if this configuration can be optimized for greater performance. The optimization involves editing the operating system configuration file that sets system parameters. System parameters are initially set according to recommendations of your operating system manuals.

You will need to check the system parameter values in your system configuration file to determine whether any of these parameters can be optimized. Increasing the value of most of these parameters reserves more memory for the system, and therefore leaves less memory for the run -time combination of user programs.

A good general rule calls for sufficient user memory to support at least 60% to 70% of the maximum load. When the system load exceeds this amount of memory, swapping and paging will occur.

Consider resetting the parameters that control the following system features.

Step 1 Check the parameter that controls:

```
the maximum number of i-node table entries
```

This parameter controls the maximum number of files in use at the same time. A file could be a directory, an ordinary file, a device (special file), or a mount point (special file).

Optimization: Calculate your expected maximum usage as follows:

```
parameter = (num_dev * 20) + (num_users * 10)
```

where num_dev is the average number of simultaneous developers and num_users is the average number of simultaneous end users.

Step 2 Check the parameter that controls:

```
the maximum number of open-file table entries
```

This parameter controls the maximum number of files that can be open at the same time. A file could be a directory, an ordinary file, a device (special file), or a mount point (special file).

Optimization: Set this parameter equal to the maximum number of i-node tables entries (found in the previous step).

Step 3 Check the parameter that controls:

```
the maximum number of entries In the process table
```

This parameter controls the maximum number of active processes that the system will support at one time.

Optimization: Calculate this parameter as follows:

```
parameter = (num_dev * 10) + (num_users; * 5)
```

where num_dev is the average number of simultaneous developers and num_users is the average number of simultaneous end users.

Step 4 Check the parameter that controls:

```
the maximum number of processes allowed for any one user
```

This parameter controls the maximum number of processes that one user can run at one time.

Optimization: Set this parameter equal to:

```
parameter = (max_proc_tab * .9)
```

where max_proc_tab is the maximum number of processes in the process table. This value was determined in the previous step.

Step 5 Check the parameter that controls:

```
the maximum size of a shared memory segment
```

The Lock Manager uses shared memory. The size of the shared memory segment used by the lock manager directory affects the granularity of the locks held. A smaller segment holds fewer lock entries, and therefore would mean an earlier promotion of locks from record to table.

This parameter controls the amount of contention when obtaining locks. The effect of the smaller segment and earlier promotion is a greater amount of contention: the chance of running into another user's lock increases.

Optimization: Calculate this parameter as follows

```
parameter = (num_dev * 4000) + (num_users * 6000)
```

where num_dev is the average number of simultaneous developers and num_users is the average number of simultaneous end users.

Step 6 To continue with the Installation procedure, return to Step 7 in Preparing to Install Unify DataServer/ELS.